Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Ivana Kolarič

Primerjava pristopov za razvoj aplikacij z ogrodjem Oracle ADF

diplomsko delo

univerzitetni študijski program računalništvo in informatika

Mentor: izr. prof. dr. Zoran Bosnić

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Oracle ADF je ogrodje, ki podpira celoten proces razvoja poslovnih aplikacij, od idejne zasnove in razvoja programske kode do namestitve. Ogrodje omogoča modularen razvoj aplikacij, temelji na zasnovi model-pogled-kontroler ter podpira ponovno uporabo napisane kode. V diplomski nalogi naj kandidatka naredi pregled pristopov in dobrih praks za razvoj aplikacij z ogrodjem Oracle ADF. Le-te naj tudi kritično ovrednoti in prikaže primer razvoja z izdelavo vzorčne aplikacije.

Izjava o avtorstvu diplomskega dela

Spodaj podpisana Ivana Kolarič, z vpisno številko **63070095**, sem avtorica diplomskega dela z naslovom:

Primerjava pristopov za razvoj aplikacij z ogrodjem Oracle ADF

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelala samostojno pod mentorstvom izr. prof. dr. Zorana Bosnića,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 3. julija 2014 Podpis avtorja:

Zahvaljujem se mentorju izr. prof. dr. Zoranu Bosniću za pomoč in nasvete pri izdelavi diplomskega dela.

Posebna zahvala gre staršem za finančno podporo tekom študija in potrpežljivost ob zaključevanju študija.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
ADF	Aplication Developement Framework	ogrodje podjetja Oracle za razvoj aplikacij
CSS	Cascading Style Sheets	stilna predloga za določanje oblike spletne strani
DML	Data Manipulation Language	jezik za manipulacijo s podatki v podatkovni bazi
JAR	Java Archive	arhiv, ki predstavlja javansko knjižnico
JSF	Java Server Faces	ogrodje spletnih aplikacij
MVC	Model-View-Controller	arhitektura, ki se uporablja za razvoj spletnih ogrodjih
SOA	Service-Oriented Architecture	storitvena arhitektura, ki dovoljuje aplikacijam izmenjavo podatkov
SQL	Structured Query Language	strukturirani povpraševalni jezik za delo s podatkovnimi zbirkami
UI	User Interface	uporabniški vmesnik
XLIFF	Localisation Interchange File Format	format, sestavljen na podlagi XML, ki standardizira prenos podatkov za lokalizacijo
XML	Extensible Markup Language	označevalni jezik, za opis strukturiranih podatkov

Povzetek

Vse večja uporaba različnih spletnih aplikacij za osebno rabo je v poslovnem svetu ustvarila zahtevo po prijaznejših in intuitivnih poslovnih aplikacijah, ki so dostopne preko spleta, upravljajo z zapletenimi poslovnimi procesi in uporabniku nudijo enostavno interakcijo. Razvoj obsežnih aplikacij je dolgotrajen in drag. V času delovanja se funkcionalnosti, ki jih vsebujejo, stalno dopolnjujejo. Zato je pomembna izbira pravega ogrodja, ki omogoča hiter in enostaven razvoj ter vsebuje komponente modernejših spletišč.

Takšno ogrodje je ogrodje Oracle ADF, ki predstavlja celotno rešitev za gradnjo poslovnih aplikacij, od ideje do namestitve v produkcijsko okolje. Poleg izbire ogrodja so pomembni dobro zastavljeni temelji aplikacije, ki omogočajo modularen razvoj aplikacij, ponovno uporabo razvitih modulov in dovolijo spremembe, ki jih zahteva projekt.

V diplomski nalogi smo predstavili različne pristope k razvoju aplikacij. Z uporabo orodja Oracle ADF smo, na podlagi vzorčne aplikacije, prikazali modularen pristop razvoja obsežnih aplikacij. Izpostavili smo nekatere probleme, ki se lahko pojavijo tekom razvoja ter katerim se bomo lažje izognili ob razvoju obsežne poslovne aplikacije.

Ključne besede: razvoj aplikacij, pristopi razvoja, Oracle ADF, poslovne aplikacije, modularen razvoj

Abstract

With the increasing use of various web applications, the demand for intuitive and user-friendly applications that are accessible via web browser is growing in the business world. Such applications must manage complex business processes and enable a simple user interaction. Because enterprise applications are constantly changing and improving, their development is very expensive. It is important to choose a framework which allows simple and rapid development, and at the same time contains modern user interface components.

Oracle ADF framework provides a complete solution for building enterprise applications from idea to post-production phase. In addition to the right choice of framework, it is important to exactly define application foundations which simplify modular development and enable reusability, and at the same time allow inevitable changes during the course of the project.

In the thesis, different approaches to develop applications with Oracle ADF framework are presented. Based on a sample application we demonstrate a modular approach which is recommended when developing large-scale enterprise applications. The thesis highlights some of the problems which may occur during development phase and could be successfully avoided when the implementation of the real enterprise application begins.

Keywords: application development, development approaches, Oracle ADF, enterprise applications, modular development

Poglavje 1

Uvod

Poslovne aplikacije so strateške aplikacije v podjetjih. So obsežne in kompleksne ter upravljajo s kritičnimi poslovnimi funkcijami. Včasih je veljalo, da je končni uporabnik potreboval navodila in tečaje za spoznavanje delovanja poslovne aplikacije. Množična družabna omrežja in spletišča so to pravilo izpodrinila. Da je aplikacija uporabniku prijazna in intuitivna, je skoraj toliko pomembno kot njeno samo delovanje. Poslovna aplikacija vsebuje veliko modulov, ki v zapletenih interakcijah sodelujejo med seboj ali med zunanjimi sistemi. Pogosto vsebuje veliko število uporabniških strani. A le dobro načrtovana poslovna aplikacija bo to kompleksnost skrila pred končnim uporabnikom [1].

Razvoj poslovnih aplikacij je dolgotrajen in drag, zato te ostanejo dolgo v uporabi. Potrebujemo ogrodje, pri katerem se razvijalec lahko osredotoči na poslovno logiko in ne na razvoj osnovnih funkcionalnosti. Ogrodje Oracle ADF (Application Development Framework) omogoča hiter, intuitiven in enostaven razvoj bogatih spletnih aplikacij. Temelji na arhitekturi MVC *(ang. Model View Controller)* in omogoča izbiro različnih tehnologij na poslovno-storitvenem ter uporabniškem nivoju. To omogoča tehnologija ADF Model, ki predstavlja model v arhitekturi MVC. Njegova naloga je abstrakcija poslovnega nivoja, s katero je omogočena šibka sklopljenost poslovnega in uporabniškega dela tudi znotraj aplikacije. Nivo kontrolerja vsebuje vezane tokove opravil, ki jih lahko, znotraj ali v drugih aplikacijah ADF, večkrat uporabimo. Na nivoju pogleda vsebuje ogrodje ADF Faces bogat nabor komponent za razvoj uporabniškega vmesnika.

Ogrodje Oracle ADF je eno izmed najbolj produktivnih ogrodij za gradnjo podatkovno vodenih aplikacij, ki so trenutno na voljo. Le z malo priprave in učenja lahko

pričnemo razvijati aplikacije, ki služijo širokemu spektru potreb [2].

Preproste aplikacije razvijamo monolitično v enem delovnem okolju, za obsežnejše aplikacije pa literatura priporoča modularen pristop, kjer različne funkcionalnosti aplikacije razvijamo v ločenih delovnih okoljih, neodvisno od ostalih modulov. Te nato poljubno združujemo v končne aplikacije. Ogrodje ADF in različne pristope za gradnjo aplikacij smo opisali v poglavju 2.

Na podlagi zalednega sistema mobilne aplikacije, smo v poglavju 3 prikazali postopek razvoja z modularnim pristopom, ki je v literaturi predlagan kot pristop za razvoj obsežnih aplikacij. Prikazali smo celoten postopek, od načrtovanja do namestitve aplikacije na strežnik.

Na koncu smo podali še prednosti in slabosti izbire različnih pristopov.

Poglavje 2

Predstavitev ogrodja Oracle ADF

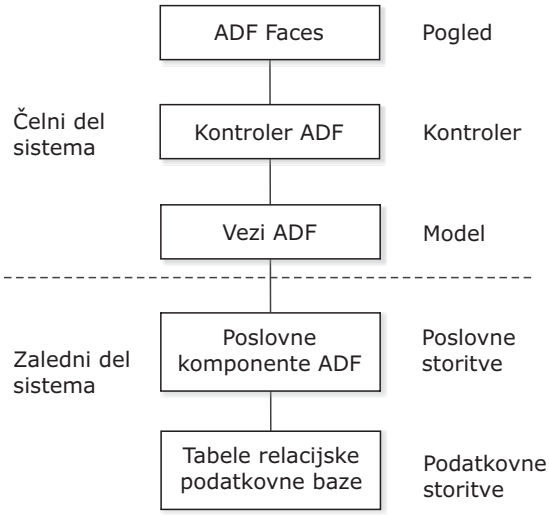
Ogrodje Oracle ADF je zgrajeno na platformi Java EE in poenostavlja razvoj poslovnih aplikacij, infrastrukturnih servisov ter skupaj z orodjem JDeveloper ponuja vizualno in deklarativno izkušnjo razvoja. Omogoča enostaven razvoj aplikacij, od ideje do namestitve v produkcijsko okolje. Razvijalec porabi manj časa za spoznavanje kompleksnosti tehnologije in se lahko bolj posveti razvoju poslovnih funkcionalnosti [3]. Temelji na arhitekturi MVC *(ang. Model View Controller)*. Zaradi vgrajene tehnologije ADF Model omogoča izbiro različnih tehnologij na poslovno storitvenem in uporabniškem nivoju.

S pomočjo knjižnic ADF ogrodje spodbuja modularen razvoj. Različne funkcionalnosti lahko razvijamo ločeno, neodvisno od ostalih modulov, in jih nato združujemo v poljubne aplikacije ADF. Knjižnice ADF so v osnovi navadne knjižnice Java z dodatnimi metapodatki, ki v orodju JDeveloper omogočajo pregled nad vsebovanimi komponentami ADF. To so lahko poslovne komponente, tokovi opravil, predloge strani [2].

2.1 Oracle Fusion Application

Obstaja veliko načinov grajenja poslovnih aplikacij z ogrodjem Oracle ADF. Za Oracle Fusion Application je uporabljena arhitektura SOA *(ang. Service-Oriented Architecture)*, ki uporablja naslednje gradnike [1]:

- **Bogat uporabniški vmesnik ADF *(ang. ADF Faces Rich Client - ADFv)***, bogat nabor komponent uporabniškega vmesnika, ki omogočajo enostavno implementacijo naprednih akcij v spletni aplikaciji.



Slika 2.1: Arhitektura Oracle Fusion Application

- **Kontroler ADF** ***(ang. ADF Controller - ADFc)***, vsebuje lastnosti kontolerja JSF *(Java Server Faces)* in ga razširi z definicijo omejenih tokov opravil, ki jih lahko uporabimo na več mestih v aplikaciji. ADF kontroler omogoča dodatno upravljanje s transakcijo, ki se lahko razteza skozi več uporabniških strani.
- **Nivo vezi ADF** ***(ang. ADF binding layer - ADFm)*** je standard, ki definira splošen model s pomočjo katerega uporabniški vmesnik komunicira s poslovno-storitvenim nivojem.
- **Poslovne komponente ADF** ***(ang. ADF Bussines components - ADFbc)*** omogočajo izboljšanje produktivnosti z deklarativnim razvojem poslovnih storitev, ki so osnovane na relacijskih podatkovnih bazah.

Arhitektura Oracle Fusion Application je prikazana na sliki 2.1.

2.2 Poslovno storitveni nivo

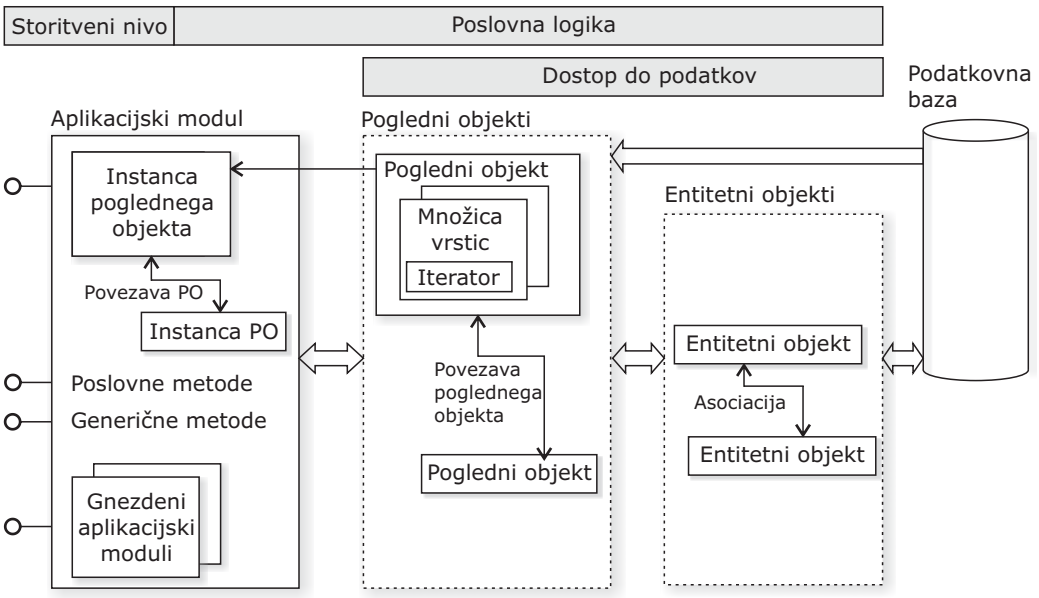
Oracle ADF podpira uporabo naslednjih poslovno storitvenih tehnologij:

- poslovne komponente ADF,
- sejna zrna EJB in entitete JPA,

- zrna Java (JavaBeans),
- spletne storitve (SOAP in REST).

2.2.1 Poslovne komponente ADF

Poslovne komponente ADF poenostavljajo razvoj in vzdrževanje kompleksnih, performančno zahtevnih in bazno usmerjenih storitev, saj se večino storitev lahko razvije deklarativno, preko čarovnikov in vizualnih urejevalnikov. Vse nastavitve, ki določajo obnašanje komponent, so shranjene v ustreznih datotekah XML, do katerih se v času izvajanja dostopa preko razredov poslovnih komponent. Te razrede lahko razširimo in jim dodamo svojo funkcionalnost [4]. Metode oziroma storitve poslovnih komponent ADF so nivoju pogleda izpostavljene skozi Model ADF, ki je podrobneje opisan v poglavju 2.3. Kot spletne storitve SOAP ali REST pa jih lahko izpostavimo zunanjim aplikacijam.



Slika 2.2: Poslovne komponente ADF in povezave med njimi. Preko aplikacijskega modula so kot storitve izpostavljene naslednje komponente: pogledni objekt z iteratorji in metode poglednega objekta (pregled, dodajanje, brisanje in spreminjanje vrstice), poslovne metode definirane v razredu aplikacijskega modula, generične metode (commit, rollback) in storitve gnezdenih aplikacijskih modulov. Dostop do podatkov se izvaja preko poglednega ali entitetnega objekta.

Elementi poslovnih komponent so: entitetni objekti, pogledni objekti, asociacije,

povezave med poglednimi objekti in aplikacijski moduli. Elementi in povezave med njimi so prikazani na sliki 2.2. V nadaljnjih poglavjih pa jih bomo podrobneje opisali.

Entitetni objekt

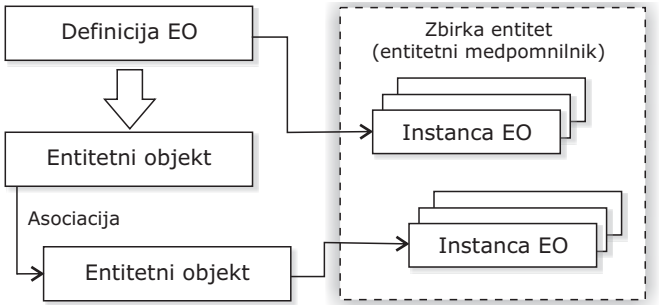
Entitetni objekti (EO) skrbijo za objektno-relacijsko preslikavo, ki omogoča predstavitev tabel relacijske podatkovne baze v objektih Java. So osnova za gradnjo poglednih objektov in skrbijo za rokovanje z baznimi podatki. Entitetni objekt predstavlja tabelo, primerek entitetnega objekta pa vrstico v tabeli. Poleg bazne tabele lahko z entitetnim objektom predstavimo tudi poglede, sinonime in materializirane poglede na podatkovni bazi. Opis entitetnega objekta je zapisan v datoteki XML. V njej so zapisane lastnosti atributov, njihovi podatkovni tipi, validatorji podatkov, primarni ključi, dolžina polj, natančnost, če gre npr. za decimalna števila in podatek o tem, ali vrednost atributa lahko spreminjamo. Poleg naštetega vsebuje entitetni atribut tudi lastnosti, na katere se sklicujemo v uporabniškem vmesniku. To so na primer namigi uporabniškega vmesnika *(ang. UI hints)*, oznaka *(ang. label)*, zaslonski namig *(ang. tooltip)* in format tipa podatkov *(ang. format type)*.

Entitetni objekt predstavljajo še naslednji Java razredi, ki jih mora vsak razvijalec poznati, ko z deklarativnim pristopom ne more razviti želene funkcionalnosti. Razredi ter povezave med njimi so prikazani na sliki 2.3.

- **Razred definicije enitete *(ang. Entity definition)*** opisuje strukturo entitetnega objekta in deluje kot predloga za definicijo entitetnega primerka v času izvajanja. Ta je generirana na podlagi definicije entitetnega objekta v datoteki XML. Privzeti razred, ki se uporablja v poslovnih komponentah ADF, je oracle.jbo.server.EntityDefImpl.

- **Entitetni objekt *(ang. Entity object)*** predstavlja primerek entitete, ta pa predstavlja vrstico v tabeli podatkovne baze. Ko iz poglednega objekta, ki je odvisen od entitetnega objekta, poženemo poizvedbo, bo ogrodje za vsako vrstico, ki je del rezultata poizvedbe, ustvarilo nov primerek entitetnega objekta. Privzeti razred je oracle.jbo.server.EntityImpl. Ta razred razširimo, kadar za manipulacijo s podatki kličemo bazno proceduro. V metodi *doDML*, ki po privzetem izvede (insert, update ali delete) stavke DML *(ang. Data Manipulation Language)* jezika SQL, definiramo klic bazne procedure.

- **Zbirka entitet** ***(ang. Entity collection)*** je entitetni medpomilnik, v katerem se pomnijo primerki določenega entitetnega objekta, ki so rezultat poizvedbe, pognane v poglednem objektu. Primerki entitetnega objekta si delijo medpomnilnik entitet, le če sodelujejo v isti transakciji. Privzeti razred je oracle.jbo.server.EntityCache [3].



Slika 2.3: gradniki entitetnega objekta

Pogledni objekt

Pogledni objekt je lahko definiran na osnovi enega ali več entitetnih objektov. Služijo za branje in pripravo podatkov, ki so del poslovne storitve ali jih prikazujemo na zaslonu. Podatke lahko v poglednih objektih, poleg entitet, dobimo kar iz poizvedbe definirane z jezikom SQL, klicem baznih procedur ali programsko, preko razredov Java. Definiramo lahko tudi statične pogledne objekte, ki podatke berejo iz datotek s končnico .properties *(ang. bundle)*. Gradniki poglednega objekta so:

- **Definicija pogleda XML** ***(ang. View definition XML)*** opisuje pogledni objekt in vsebuje naslednje elemente:
  - poizvedbo SQL,
  - vezne spremenljivke, ki jih uporabimo kot parametre v poizvedbi ali poglednih kriterijih,
  - definicijo atributov, ki jih poizvedba vrne,
  - informacijo o entitetah, na podlagi katerih je ustvarjen pogledni objekt,

  - definicije seznamov vrednosti *(ang. list of values LOV)* in poglednih kriterijev *(ang. View criteria)*,
  - dostop do ostalih poglednih objektov *(ang. View accessor)*; uporabljajo se pri določanju seznamov vrednosti na atributih ter validacijah,
  - pravila poslovnega modela,

- **Definicija pogleda** ***(ang. View definition)*** je razred Java, ki v času izvajanja predstavlja definicijo datoteke XML. Privzeti razred je oracle.jbo.server.ViewDefImpl.

- **Pogledni objekt** ***(ang. View object)*** predstavlja primerek poglednega objekta ter upravlja s poizvedbami v času izvajanja. Privzeti razred je oracle.jbo.server.ViewObjectImpl.

Ogrodje uporablja naslednje razrede, za upravljanje z rezultati poizvedb.

- **Vrstica** ***(ang. Row)*** predstavlja vrstico, ki je del rezultata poizvedbe.

- **Množica vrstic** ***(ang. Row set)***, upravlja z vrnjenimi podatki poizvedbe. Množica vrstic vsebuje vrednosti veznih spremenljivk, s katerimi je bila pognana poizvedba in iteratorje za pregled vrstic. Pogledni objekt ima lahko več množic vrstic, ki se med seboj razlikujejo po vrednosti veznih spremenljivk.

- **Iterator množice vrstic** ***(ang. Row set iterator)*** je vsebovan v množici vrstic in omogoča iteracijo po rezultatih poizvedbe. V posamezni množici je lahko več iteratorjev. Primer sta iterator za prikaz tabele na zaslonu in iterator za pregled vrstic v upravljalnem zrnu, kjer izvajamo dodatno računanje nad podatki.

- **Zbirka poizvedb** ***(ang. Query collection)*** hrani rezultate poizvedbe. Primerek poglednega objekta ima lahko več zbirk poizvedb. Ustvarijo se glede na vrednosti veznih spremenljivk v množici vrstic. Če več množic vrstic uporablja iste vrednosti parametrov za poizvedbo, bo ogrodje uporabilo isto zbirko za hrambo rezultatov. Zbirka poizvedb hrani dejanske vrstice, medtem ko jih množica vrstic ne.

Asociacije in povezave poglednih objektov

Asociacije definirajo povezavo med definicijami entitetnih objektov. Predstavljamo si jih lahko kot tuje ključe na relacijskih podatkovnih bazah.

Pogledne objekte med seboj povezujemo s poglednimi povezavami. Povezujejo nadrejene *(ang. master)* in podrejene *(ang. detail)* pogledne objekte, kjer je nabor vrstic podrejenega objekta odvisen od izbrane vrstice v nadrejenem poglednem objektu.

Aplikacijski modul ADF

Aplikacijski modul ovije primerke poglednih objektov in metode poslovnih storitev, ki so potrebne za izvajanje neke enote dela. Običajno aplikacijski modul predstavlja večjo funkcionalnost v aplikaciji.

Vsak aplikacijski modul ima svoj kontekst transakcije, ki uporablja svojo povezavo na bazo. To pomeni, da so vsi pogledni objekti in metode, znotraj modula, del iste transakcije. Aplikacijske module se lahko gnezdi, pri tem pa vsi uporabljajo transakcijo, definirano v korenskem modulu *(ang. root application module)*. Z gnezdenjem modulov pogostokrat prihranimo na številu povezav na bazo [1]. Aplikacijski modul definirajo:

- **Datoteka XML metapodatkov** ***(ang. Aplication module XML)*** vsebuje opis podatkovnega modela in metod, ki so izpostavljene odjemalcu. V opis podatkovnega modela spadajo pogledni objekti in pripadajoče pogledne kriterije, nadrejeni in podrejeni pogledni objekti, ki so med seboj povezani z poglednimi povezavami ter gnezdeni aplikacijski moduli.

- **Definicija aplikacijskega modula** ***(ang. Aplication module definition)*** je razred Java, ki skrbi za predstavitev lastnosti, definiranih v datoteki XML, v času izvajanja.

- **Implementacija aplikacijskega modula** ***(ang. Aplication module implementation)*** predstavlja primerek aplikacijskega modula, ki se generira ob izvajanju z namenom, da streže prejetim zahtevam. Kadar je potrebno implementirati dodatno poslovno logiko, se to stori v razširitvenem razredu tega razreda. Privzeti razred je oracle.jbo.server.ApplicationModuleImpl.

- **Datoteka bc4j.xcfg** vsebuje podatke, ki določajo obnašanje aplikacijskega modula v času izvajanja. To so na primer: podatki o povezavi na bazo, velikost

bazena aplikacijskih modulov *(ang. Application Module Pooling)*, življenjska doba aplikacijskega modula, interval, ki določa kdaj se neaktivni moduli sprostijo nazaj v bazen aplikacijskih modulov, itd.

Aktivacija in pasivacija aplikacijskih modulov

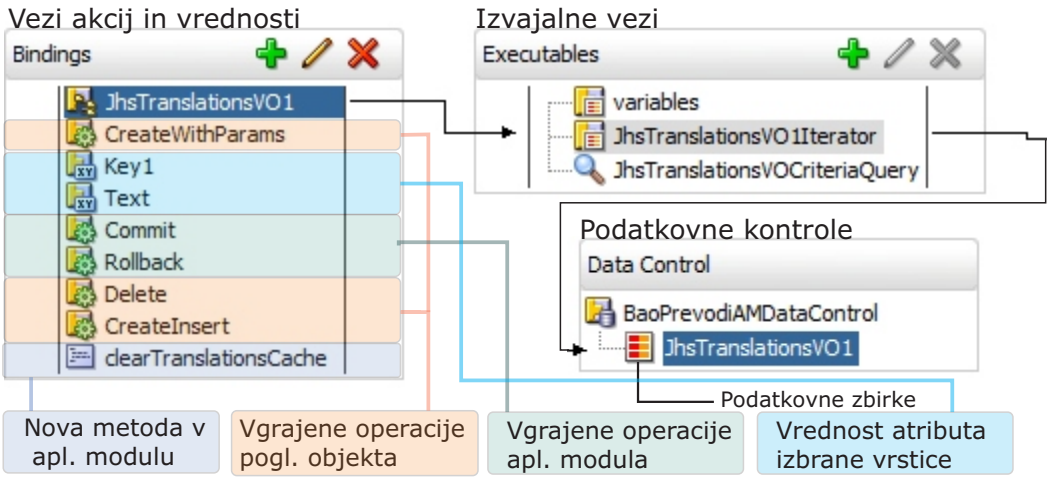
Kadar imamo v aplikaciji prijavljenih več uporabnikov, kot je na voljo primerkov v bazenu aplikacijskih modulov, pride do pasivacije *(ang. passivation)* in aktivacije *(ang. activation)*. Ko nov uporabnik zahteva primerek aplikacijskega modula in v bazenu aplikacijskih modulov ni prostega primerka, pride do pasivacije. Pri pasivaciji se vzame primerek aplikacijskega modula, ki je največ časa neaktiven. Njegovo stanje se shrani ali na bazo ali v datoteko na strežniku. Shranijo se podatki o seji, začasni podatki ter poizvedbe, ki jih je uporabnik pognal. Nato se primerek uporabi za streženje novemu uporabniku. Ko prvi uporabnik ponovno zahteva operacijo aplikacijskega modula in pridobi primerek, pride do aktivacije. Ob aktivaciji se ustrezno napolnijo začasni podatki, poleg tega pa se še enkrat poženejo poizvedbe poglednih objektov. Aktivacija aplikacijskega modula upočasni sistem, zato je potrebno paziti, da ustrezno nastavimo velikost bazena aplkacijskih modulov [5].

Prehodni atributi

Na entitetnih in poglednih objektih lahko definiramo prehodne atribute *(ang. transitive attributes)*. Atributi niso preslikava določenega stolpca na bazi. Njihovo vrednost računamo programsko, v razredu entitetnega objekta, ali pa operacijo definiramo z jezikom Groovy preko deklarativnega urejevalnika. Prehodne atribute lahko uporabimo za operacije, kot so štetje elementov podrejenega objekta ali računanje povprečja vrednosti določenega stolpca. Uporabni so tudi za shranjevanje začasnih podatkov, ki določajo stanje in delovanje določene funkcionalnosti. Takrat moramo paziti, da na prehodnih atributih omogočimo pasivacijo. Ob pasivaciji in aktivaciji aplikacijskega modula se bo vrednost shranila, sicer se izbriše in delovanje aplikacije ne bo pravilno.

2.3 Model ADF

Osrednji del ogrodja Oracle ADF predstavlja Model ADF. Sestavljajo ga podatkovne kontrole *(ang. Data Control)* in podatkovne vezi *(ang. Data Bindings)*. To so datoteke



Slika 2.4: Vsebovalnik vezi. Na sliki je prikazana povezava med elementi vsebovalnika vezi in operacijami poslovnih komponent.

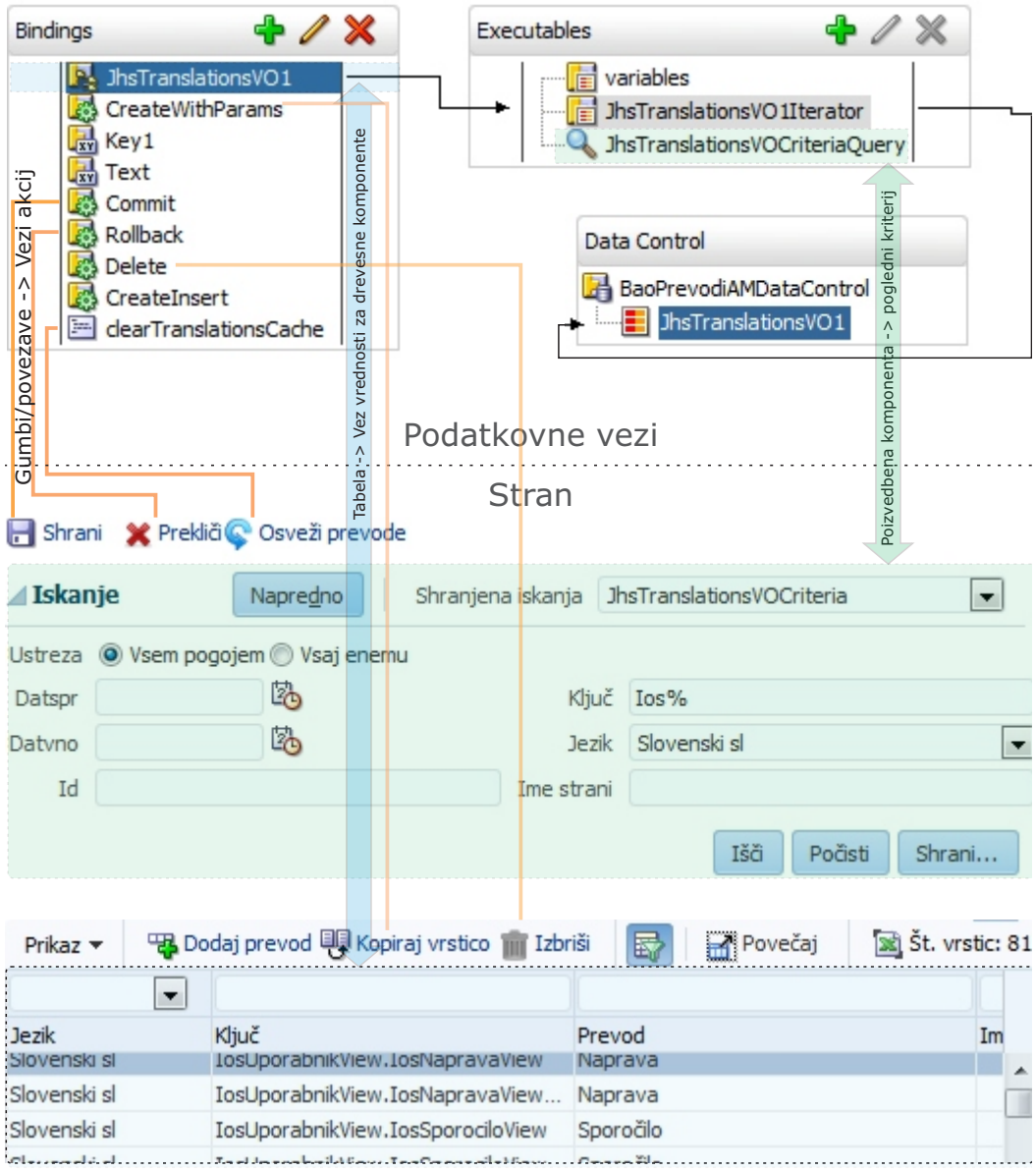
XML, v katerih so zapisane lastnosti storitev, lastnosti podatkovnih struktur, metod in podatkovnih tipov poslovnih storitev.

Podatkovne kontrole so zapisane v datoteki DataControls.xml. V njej je zapisano, katere podatkovne vezi pripadajo določeni strani, fragmentu strani ali toku opravil. Pri uporabi poslovnih komponent ADF podatkovne kontrole omogočajo dostop do aplikacijskih modulov, njihovih metod ter vsebovanih poglednih objektov.

Podatkovne vezi so shranjene v vsebovalniku podatkovnih vezi *(ang. binding container)*. Vsaka stran, fragment strani ali akcija znotraj toka opravil ima svoj vsebovalnik podatkovnih vezi. Ta se ustvari samodejno, ko na primer na stran dodamo element podatkovne kontrole.

Poznamo tri vrste objektov vezi:

- **Izvajalne vezi**, med katere spadajo vezi iteratorjev za pomikanje po rezultatih v tabeli in vezi, ki skrbijo za izvajanje nalog v ozadju. To so npr. vezi za upravljanje s poizvedbami.
- **Vezi vrednosti** skrbijo za prikaz podatkov v komponentah uporabniškega vmesnika. Vezi vrednosti so vezi za prikaz drevesnih tabel, seznamov, atributov, itd.
- **Vezi akcij** se uporabljajo za vezanje akcij na komponente uporabniškega vmesnika, kot so gumbi in povezave.



Slika 2.5: Vsebovalnik vezi. Prikazana je povezava med komponentami uporabniškega vmesnika in elementi vsebovalnika vezi.

Njihova razporeditev v vsebovalniku vezi in povezava s poslovnimi komponentami je prikazana na sliki 2.4, povezava s komponentami uporabniškega vmesnika pa na sliki 2.5.

Podatkovne kontrole omogočajo abstrakcijo implementacije poslovnih procesov, zato lahko poslovne procese implementiramo v različnih tehnologijah, ki smo jih navedli v poglavju 2.2. V času izvajanja se iz podatkovnih kontrol prebere, katera podatkovna vez pripada strani ali toku opravil, ki ga uporabnik zahteva. Nato se vzpostavi dvosmerna povezava med uporabniškim vmesnikom in poslovnimi storitvami aplikacije [4].

Na strani se na vrednosti vezi sklicujemo preko izraznega jezika *(ang. expression language)*. V kodi je prikazan sklic na vez akcije v komponenti uporabniškega vmesnika *<af:link>*.

```
<af:link actionListener="#{bindings.Delete.execute}"
         disabled="#{!bindings.Delete.enabled}"
         id="cil2" immediate="true" text="Delete" />
```

2.3.1 Knjižnica vezi ADF (ADF binding API)

Velikokrat se srečamo s problemom, ko podatkovne kontrole in vezi niso dovolj za implementacijo določene funkcionalnosti. Včasih moramo izvesti kodo v upravljalnem zrnu strani, preden pokličemo metodo iz podatkovnih storitev. Ali pa želimo rokovati s podatki v razredu Java, kjer nimamo podatkovnih kontrol in vezi.

Kadar želimo izvajati akcije poslovnih storitev v upravljalnem zrnu strani, moramo to storiti preko knjižnice vezi ADF *(ang. ADF binding API)*. Pomembno je, da se operacijo poslovnih storitev najprej definira v vezeh, kot vez akcije, in nato preko knjižnice pokliče v upravljalnem zrnu. Najbolj pogosta napaka razvijalcev je ta, da preko vezi dobijo razred aplikacijskega modula in nato izvedejo njegovo metodo. Pri tem pristopu niti ni potrebno, da je metoda aplikacijskega modula izpostavljena odjemalcu. Kljub temu, da aplikacija deluje, je priporočljivo, da razvijalec ne preskoči nivo vezi, saj s tem onemogoči šibko sklopljenost nivojev pogleda ter modela. Izgubi se tudi način lovljenja napak oziroma podatek o rokovalniku z napakami *(ang. error handler)*, ki je definiran v podatkovni kontroli. Kadar želimo vseeno priklicati metodo poslovnih komponent na način, kjer metoda ni definirana v vezeh, moramo implementirati klic metode *reportException()* razreda *DCBindingContainer*. S to metodo določimo, kateri vsebovalnik vezi naj se uporabi pri rokovanju z napako. Če tega ne storimo, se napaka morda ne bo

prikazala uporabniku na strani [3]. Primer programskega klica operacije, definirane v vezeh strani, je prikazan v podpoglavju razvoja modula za varnost 3.9.1.

Če razvijamo funkcionalnost, ki v ozadju nima strani, podatkovnih kontrol in vezi, moramo primerek aplikacijskega modula klicati programsko. Takrat je pomembno, da primerek, po zaključku neke operacije, sprostimo nazaj v bazen aplikacijskih modulov in s tem omogočimo uporabo istega primerka drugim uporabnikom. V nasprotnem primeru lahko pride do uhajanja pomnilnika *(ang. memory leak)*. Primer bo prikazan v poglavju branja prevodov iz podatkovne baze 3.8.1.

2.4 Kontrolni nivo

V ogrodju ADF lahko kontrolni nivo implementiramo s kontrolerjem JSF *Java Server Faces* ali z uporabo kontolerja ADF, ki temelji na tehnologiji JSF. Ena glavnih prednosti kontrolerja ADF je ta, da razdeli tok aplikacije na več manjših, omejenih tokov opravil *(ang. bounded task flows)*. Omejeni tokovi opravil so ena bistvenih komponent ogrodja ADF. Omogočajo grajenje manjših enot, ki se lahko uporabijo samostojno ali znotraj verige tokov, ki jo določa tok aplikacije [6].

Tok opravil ADF, poleg strani in navigacije med njimi, vsebuje tudi uporabniku nevidne komponente. To so klici metod in odločitvene točke *(ang. routers)*, ki na podlagi podanih vrednosti navigirajo na prikaz strani ali izvajanje metod [7]. V toku opravil lahko izvajamo metode, ki so implementirane v upravljalnih zrnih Java ali metode poslovnih procesov preko vezi Modela ADF.

Poleg omejenih tokov opravil poznamo tudi neomejen tok opravil. Primer slednjega je datoteka adfc-config.xml. Aplikacija ima lahko samo en neomejen tok, ta pa lahko vsebuje več omejenih tokov opravil, strani, aktivnosti, definicij zrn in ostalih elementov toka opravil. Kadar aplikacijo razvijamo modularno, se v vsakem modulu ustvari datoteka adfc-config.xml. Te se v času izvajanja združijo v eno. Pri modularnem načinu razvoja aplikacije nam tako ni potrebno definirati istega upravljalnega zrna, strani ali akcije na več mestih.

2.4.1 Primerjava z Java Server Faces

Navigacijski model Java Server Faces (JSF) je zgrajen na podlagi navigacijskih pravil, shranjenih v datoteki faces-config.xml. Pravila, na podlagi izida *(ang. outcome)*,

ki ga povzročajo komponente uporabniškega vmesnika, odločajo o prikazu naslednje strani. ADF za navigacijo uporablja tok opravil. Datoteka faces-config.xml se pri uporabi kontrolerja ADF še vedno uporablja za določanje validacijskih pravil, pretvornikov, poslušalcev faz življenjskega cikla *(ang. lifecycle phase listners)*, definicijo lastnih komponent in globalnih virov prevodov [3]. Še ena prednost toka opravil ADF je ta, da vsebuje skupni pomnilnik *(ang. shared memory scope)*. Primer je pomnilnik toka opravil *(ang. page flow scope)*, ki omogoča prenašanje podatkov med aktivnostmi znotraj toka opravil. V JSF, razen obsega seje *(ang. session scope)*, tega ni [8].

Zgornja primerjava je narejena na podlagi JSF 2.0, na kateri temelji kontroler ADF. V času pisanja je zunaj že nova različica JSF 2.2, ki po izgledu toka opravil ADF in spletnega toka ogrodja Spring *(ang. Spring web flow)*, vsebuje tokove imenovane Faces flow [9]. Trenutna različica ogrodja Oracle ADF 12 ne podpira JSF 2.2.

2.5 Nivo pogleda

ADF podpira naslednje tehnologije na nivoju pogleda [1]:

- **ADF Faces,**
- **Apache MyFaces Trinidad**, odprtokodno spletno ogrodje, ki temelji na JSF in je osnova za komponente ADF Faces,
- **JSF, jedro Java Server Faces,**
- **ADF Mobile**, podpira spletne in vgrajene *(ang. native)* brskalnike mobilnih naprav,
- **Microsoft Excel**, kot čelni del sistema za poslovne komponente ADF.

2.5.1 ADF Faces

ADF Faces je zbirka več kot 150 komponent, ki so zgrajene na osnovi ogrodja Java Server Faces in vsebujejo funkcionalnosti Ajax *(ang. Asynchronous JavaScript and XML)*. Kjub temu je razvijalcu kompleksnost jezika JavaScript skrita, saj se funkcionalnost komponent razvija deklarativno. V primerih, ko želi razvijalec dodati svojo funckionalnost na strani odjemalca, lahko to stori z uporabo ustreznih metod JavaScript, ki jih ogrodje

ADF ponuja znotraj ogrodja odjemalca *(ang. client-side framework)*. Za sklicevanje na večino komponent se uporablja razred JavaScript *AdfUIComponent* in njegove podrazrede. Primerek tega razreda je predstavitev strežniške komponente na strani odjemalca [10]. ADF Faces delimo v naslednje kategorije:

- **Komponente postavitve *(ang. Layout Components)*:** Določajo postavitev strani in vsebujejo interaktivne komponente, katerih vsebino lahko prikazujemo ali skrivamo. Poleg standardnih postavitvenih komponent spadajo sem še meniji in vsebovalniki orodnih vrstic, sekundarna okna, kot je na primer pojavno okno *(ang. popup)*, v katerem ADF faces omogoča prikazovnje celotnega toka opravil.

- **Pogledi podatkov *(ang. Data views)***: Komponente za prikaz podatkov, kot so tabele in drevesne tabele, ki omogočajo razvrščanje in filtriranje podatkov. Podatke lahko prikazujemo tudi z bolj kompleksnimi komponentami, kot so: grafi, diagrami, pivot tabele, časovnice ter geografske in tematske karte.

- **Poizvedbene komponente *(ang. Query components)***: Na podlagi kriterija pogleda nekega poglednega objekta se s pomočjo poizvedbene komponente ustvari iskalnik. Komponenta omogoča osnovno in napredno iskanje, dodajanje iskalnih atributov, dinamično izbiro poglednega kriterija ter shranjevanje in personalizacijo iskalnih pogojev. V naprednem iskanju lahko uporabnik sam določi relacijo *(npr. strartwith ali contains)* za iskanje po določenem atributu. Poizvedbena komponenta, kot jo vidi uporabnik, je prikazana na sliki 2.5 v poglavju 2.3.

ADF Faces vsebuje še komponente za sporočila in pomoči, splošne kontrole, kot so navigacijske komponente in komponente za prikaz slik in ikon, ter operacije, ki s komponentami delujejo in omogočajo implementacijo dodatnih funkcionalnosti, kot so: akcije *primi in spusti (ang. drag and drop)*, validacije in dogodkovni poslušalci *(ang. event listeners)*.

Pomemben del so tudi predloge izgleda *(ang. Page Templates)*, ki poenostavljajo razvoj posameznih strani.

S pomočjo orodja za oblikovanje *(ang. ADF Skinning)*, lahko spreminjamo izgled komponent ogrodja ADF Faces. Obliko določamo v datoteki CSS ogrodja ADF, ki vsebuje posebno obliko jezika CSS *(ang. Cascading Style Sheets)*. Za vsako komponento obstaja izbirnik *(ang. selector)*, s pomočjo katerega to komponento oblikujemo.

Z ogrodjem ADF gradimo strani *(ang. pages)*, ki imajo končnico *.jsf* ali *.jspx* ter fragmente strani *(ang. fragments)*, ki imajo končnico *.jsff*. Fragmente strani se uporablja v vezanem toku opravil. Strani uporabljamo za prikaz glavne strani ali za odpiranje toka opravil v pojavnem oknu.

2.6 Lokalizacija

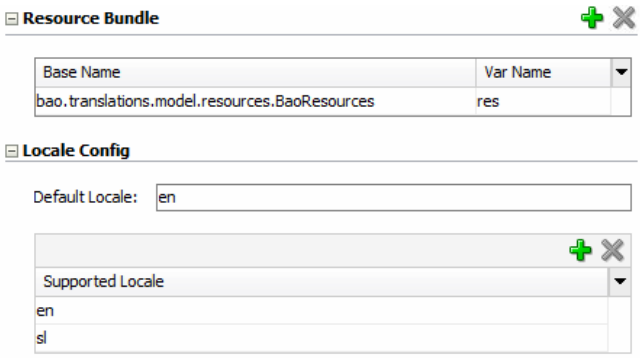
Ogrodje ADF in orodje JDeveloper omogočata avtomatizacijo lokalizacije. Prevode se lahko definira na nivoju atributov entitet in poglednih objektov, ki so del poslovnih komponent ADF. Atribut entitete "UporabniskoIme" bo na vsaki strani uporabniškega vmesnika preveden v niz "Uporabniško ime", razen če ni drugače določeno. Ogrodje bo pregledalo prevode za ključe na nivoju poglednega objekta in šele nato iskalo prevode na nivoju entitete. Prevodi in ključi so lahko zapisani v datoteki s končnico *.properties*, v datoteki XLIFF (XML format) ter v razredu Java *ListResourceBundle.java*. V orodju JDeveloper se prevode ureja deklarativno, z urejanjem polja "UI Hints" v atributu entitete ali poglednega objekta. Razvijalec v polje zapiše želeni niz in avtomatsko se ustvari ena od naštetih datotek, ki vsebuje ključ prevoda ter prevod. V definiciji entitete ali poglednega objekta se doda element "LABEL", ki vsebuje ključ prevoda:

```
<ViewAttribute Name="Naslov" IsNotNull="true" PrecisionRule="true"
EntityAttrName="Naslov" EntityUsage="IosAkcija" AliasName="NASLOV">
   <Properties>
     <SchemaBasedProperties>
        <LABEL ResId="iaug.akcija.model.view.IosAkcijaVO.Naslov"/>
     </SchemaBasedProperties>
   </Properties>
 </ViewAttribute>
```

Ključ je sestavljen na podlagi paketa, imena poglednega objekta ali entitete in imena atributa. V entiteto in pogledni objekt se zapiše še pot do datoteke, v kateri so shranjeni pari ključev in prevodov:

```
<ResourceBundle>
   <PropertiesBundle
       PropertiesFile="bao.translations.model.resources.BaoResources"/>
</ResourceBundle>
```

Za prevajanje oznak polj, ki jih ustvarimo na strani in niso del poslovnih objektov, moramo v datoteki *faces-config.xml* definirati vir prevodov. V datoteki določimo spremenljivko, s katero se bomo sklicevali na razred s prevodi (prikazano na sliki 2.6). Na



Slika 2.6: Nastavitev datoteke resource budnle v faces-config.xml

elementih uporabniškega vmesnika dostopamo do prevodov z imenom spremenljivke in imenom ključa. Primer: $\#\{res.GumbShrani\}$, kjer je *res* ime spremenljivke *GumbShrani* pa ime ključa.

2.7 ADF Essentials

Oracle ADF je bil do jeseni 2012 na voljo le uporabnikom, ki so bili pripravljeni plačevati drage licence v primeru namestitve aplikacije v produkcijsko okolje. Kljub temu, da so orodja za razvoj aplikacij zastonj, je to predstavljalo veliko omejitev pri izbiri ogrodja za razvoj aplikacij.

Oracle je, v upanju da razširi ogrodje ADF med razvijalce, izdal brezplačno različico ogrodja ADF. To je ADF Essentials, ki vsebuje vse funkcionalnosti za razvoj podatkovno vodenih aplikacij in omogoča uporabo ogrodja ADF izven celotnega oraclovega sklada *(ang. Oracle stack)*. Iz tega razloga ne vsebuje vseh funkcionalnosti ogrodja ADF, saj so nekatere tesno povezane z uporabo strežnika Weblogic. Nekatere so:

- ADF Security, ki skrbi za varnosti v aplikaciji,
- MDS *(Metadata Services)*, ogrodje za prilagoditev aplikacije glede na posamezno namestitev ali celo posameznemu uporabniku. Za uporabnika lahko shrani različne postavitve komponent, kot je na primer razporeditev stolpcev v tabeli [8],

- EM *(Enterprise Manager)*, orodje za konfiguracijo strežnika in aplikacij, spremljanje porabe virov, ter upravljanje opravil.

Oracle je ADF Essentials izdal z namenom, da večjemu številu razvijalcem predstavi razvojne koncepte ogrodja ADF in v upanju, da bodo razvijalci lahko sami spoznali, kako bogato in produktivno je to ogorodje. ADF Essentials aplikacije lahko namestimo na oraclova strežnika Weblogic ali Glassfish. Ker pa ni nobenih omjitev z licencami, lahko aplikacije namestimo tudi na ostale strežnike Java EE [11, 2].

Oracle ADF in ADF Essentials nista odprtokodna. Razvijalci, ki imajo s podjetjem Oracle pogodbo o zagotvaljanju podpore, lahko zahtevajo kopijo kode ogrodja ADF. Ta podpora pa seveda ni brezplačna [12].

2.8 Pristopi razvoja aplikacij z ogrodjem ADF

Kot večina orodij IDE, tudi orodje JDeveloper deluje na podlagi koncepta delovni prostor *(ang. workspace)*, ki ga poimenuje kar aplikacija *(ang. application)*. Delovni prostor oziroma aplikacija vsebuje enega ali več projektov. Pri aplikacijah Fusion Web sta to projekt modela ter projekt pogleda in kontrolerja.

Pri razvoju spletnih aplikacij je odločanje o arhitekturi prva stvar, ki ji je potrebno nameniti precejšnji delež razvojnega časa. Z ogrodjem ADF je za razvoj aplikacij mogočih več pristopov. Aplikacijo lahko razvijamo v enem samem delovnem prostoru ali pa jih uporabimo več. Z orodjem JDeveloper in knjižnicami ADF lahko več aplikacij ADF združimo v eno glavno. Izbira pristopa razvoja je odvisna od velikosti aplikacije, potrebi po deljenju kode med posameznimi aplikacijami, načinu nalaganja aplikacije na strežnik *(ang. deploy)*, ciklov izdajanja novih različic ter števila razvijalcev na projektu. V naslednjih poglavjih bomo različne pristope podrobneje opisali.

2.8.1 Monolitične aplikacije

Monolitično aplikacijo sestavlja eno delovno okolje *(ang. workspace)*, ki vsebuje enega ali več projektov modela in pogleda. Aplikacija je zapakirana v datoteko *ear* in na strežniku teče kot en proces.

Najpreprostejši pristop k razvoju aplikacije je ustvarjanje delovnega okolja, ki vsebuje en projekt modela in en projekt pogleda. Projekt modela vsebuje entitetne objekte,

pogledne objekte in enega ali več aplikacijskih modulov. Projekt pogleda vsebuje strani JSF, tokove opravil, fragmente strani ter podatkovne kontrole in vezi. Ta pristop razvijanja aplikacij je preprost in primeren za aplikacije, ki jih razvija en razvijalec in kjer ni potrebe po delitvi funkcionalnosti med ostale aplikacije. Slabost tega pristopa je, da posameznih delov aplikacije ne moremo večkrat uporabiti.

V naslednjem pristopu delovno okolje vsebuje več projektov modela in en sam projekt pogleda. Pri tem pristopu je potrebno paziti, da se poslovne funkcionalnosti razdelijo v smiselne projekte modela. Paziti je potrebno na odvisnosti med posameznimi projekti ter odstraniti morebitne ciklične odvisnosti. Pristop je primeren za razvoj manjših in srednje obsežnih aplikacij, kjer je število uporabniških strani majhno (manj kot petdeset) in kjer je interakcija med ostalimi aplikacijami majhna.

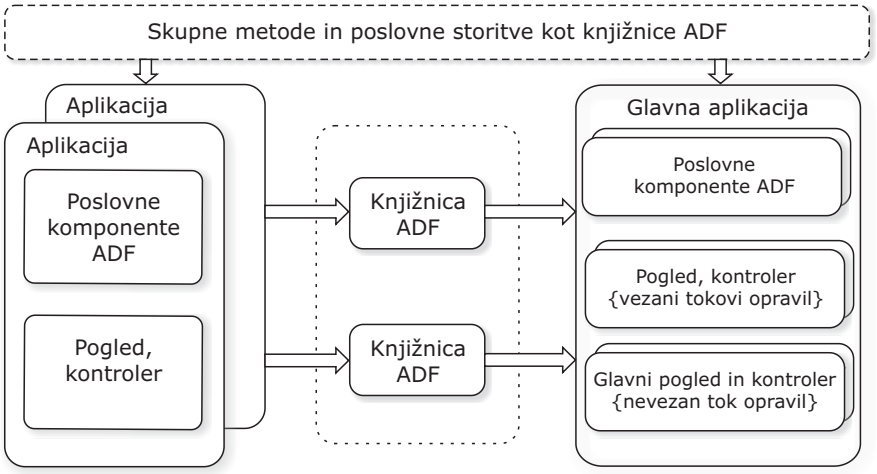
Za aplikacije, ki vsebujejo več kot petdeset uporabniških strani, se uporablja pristop, kjer delovni prostor, poleg več projektov modela, vsebuje tudi več projektov pogleda. En izmed projektov pogleda je glavni *(ang. master)* in se uporablja za združitev funkcionalnosti vseh ostalih projektov. Vsak projekt pogleda vsebuje vezane tokove opravil, ki predstavljajo del neke funkcionalnosti. Glavni projekt vsebuje konfiguracijske datoteke, skripte za gradnjo *(ang. build)* aplikacije in informacijo o odvisnosti od ostalih projektov. Število projektov v delovnem okolju je odvisno od modularnosti, ki jo razvijalec želi doseči.

2.8.2 Modularne aplikacije

Pristop razvoja monolitičnih aplikacij hitro propade, kadar želimo razviti aplikacijo, sestavljeno iz neodvisnih modulov, ki jih lahko večkrat uporabimo in kadar želimo v eni aplikaciji preko knjižnice uporabljati storitve druge aplikacije. Podjetje lahko določene funkcionalnosti uporabi v več različnih aplikacijah. Rešitev za ta pristop je šibka sklopljenost poslovnih procesov. Poslovne funkcionalnosti se lahko razdeli v več podsistemov oziroma modulov, ki jih lahko poganjamo neodvisno od končne aplikacije [3]. Diagram je prikazan na sliki 2.7. Prednosti tega pristopa so:

- izboljšana modularnost sistema,
- vzdrževanje postane enostavnejše; vsak modul lahko neodvisno od ostalih spremenimo in naložimo na strežnik, ne da bi pri tem onemogočili delovanje celotne aplikacije,

- hitrejši čas razvoja; testiranje novih modulov je neodvisno od končne aplikacije,
- module lahko uporabimo večkrat in v različnih aplikacijah.



Slika 2.7: Diagram razvoja poslovne aplikacije. Več manjših projektov modela in pogleda se združi v glavni aplikaciji.

Tak pristop prinaša tudi nekaj slabosti. Orodje JDeveloper ne zna najboljše rokovati z odvisnostmi med posameznimi moduli in je za to potrebno nekaj discipline med razvijalci. Ker imamo več aplikacij, je tudi nadzor različic kode otežen, saj mora biti koda vsakega modula oziroma aplikacije neodvisno verzionirana od kode ostalih modulov. Pomembno je, da granulacija aplikacije ni predrobna, saj se lahko zgodi, da razvijalec porabi preveč časa za urejanje kode in knjižnic, namesto za razvijanje novih funkcionalnosti.

Odvisnost modulov in izogibanje ciklom

Največji izziv pri razdeljevanju projekta v smiselne module predstavlja določanje odvisnosti med posameznimi moduli in izogibanje cikličnim odvisnostim. Pri razdeljevanju se lahko uporabi osnovno pravilo razdeljevanja monolitičnih aplikacij v več projektov [3]:

1. Najprej določimo skupne storitve in pomožne metode aplikacije. Vse te postavimo v projekt, kjer hranimo splošno kodo.

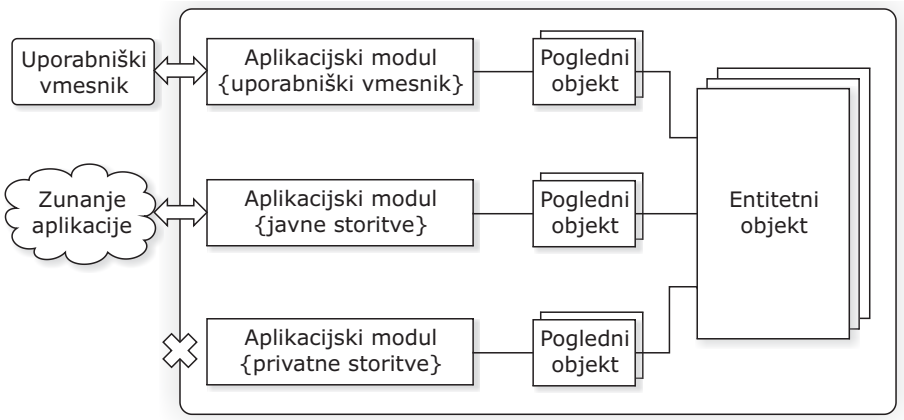
2. Določimo logično neodvisne poslovne module in za njih zgradimo projekte modela. Projekti so lahko odvisni le od projekta splošne kode.

3. Definiramo podrobnejši nivo poslovnih modulov in za njih zgradimo projekte modela. Pri tem je potrebno paziti, da so moduli odvisni le od tistih definiranih na višjem nivoju (moduli, ki smo jih definirali v prejšnjih korakih) in ne na istem. Ta korak nadaljujemo, dokler vseh uporabniških primerov ne razdelimo v ustrezne module. Žal za ta korak ne obstaja nobeno učinkovito in hitro pravilo. Zato se med razvojem lahko zgodi, da bo obstoječe module potrebno spremeniti, da se izognemo cikličnim odvisnostim.

Ko ustvarjamo aplikacije in projekte znotraj njih, moramo vsakemu projektu definirati enolično ime paketa. To je pomembno, saj ogrodje ADF za vsako delovno okolje ustvari konfiguracijske datoteke. Z enoličnimi imeni paketov poskrbimo, da se te datoteke ne povozijo, ko module združimo v glavno aplikacijo.

2.8.3 Granulacija posamezne aplikacije

Ravno tako kot pri monolitičnem pristopu lahko tudi pri modularnem pristopu posamezna delovna okolja vsebujejo več projektov modela. Ti projekti so organizirani glede na to, kako si med seboj delijo kodo. Aplikacija lahko vsebuje dva projekta modela za



Slika 2.8: Prikaz diagrama razdelitve projekta modela nekega podsistema oz. aplikacije.

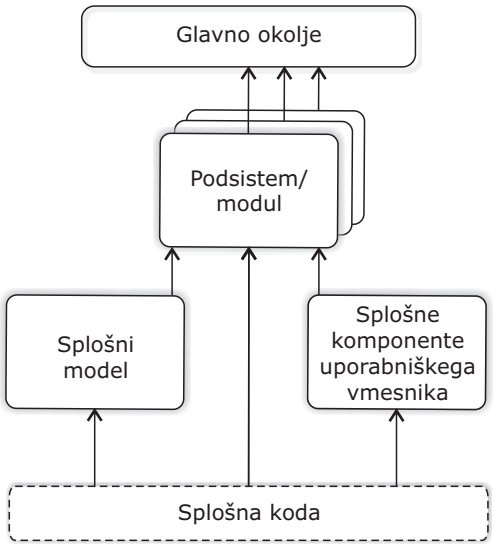
neko funkcionalnost. Vsak projekt modela vsebuje po en aplikacijski modul. Prvi modul

vsebuje zasebne metode znotraj projekta. Lahko je vsebovan v aplikacijskem modulu z javnimi metodami, ki so preko storitev izpostavljene zunanjim aplikacijam. Po želji lahko dodamo še modul, ki vsebuje metode, potrebne za prikaz uporabniškega vmesnika. Slednji vsebuje aplikacijski modul z javnimi metodami. Na sliki 2.8 je prikazano omenjeno gnezdenje aplikacijskih modulov [3]. Tak pristop je seveda primeren, v kolikor so posamezni moduli dovolj obsežni, da jih je smiselno ločevati v podprojekte.

2.8.4 Razvoj poslovnih aplikacij

Modularen pristop razvoja aplikacij je primeren za obsežne poslovne aplikacije, saj vsebujejo veliko datotek in kode. Da preprečimo kaos, je potrebno aplikacijo razdeliti na manjše, obvladljive dele. Orodje JDeveloper in ogrodje ADF to omogočata s koncepti delovnega prostora *(ang. workspace)* in knjižnicami ADF [1].

Namesto ene obsežne aplikacije moramo razviti več manjših. Razdelimo jih po naslednjih sklopih, ki so prikazani na sliki 2.9:



Slika 2.9: struktura aplikacije

- **Aplikacija s splošno kodo** vsebuje najbolj pogosto uporabljene metode, ki so skupne celotnemu sistemu. Vsebuje tudi razširitvene razrede ogrodja za npr.: pogledne objekte, entitetne objekte in aplikacijske module ADF.

- **Aplikacija za skupne komponente uporabniškega vmesnika** vsebuje predloge izgleda, predloge toka opravil, ter posebno datoteko CSS za oblikovanje komponent ADF *(ang. ADF Skinning)*.

- **Aplikacija za splošne poslovne komponente** vsebuje entitete celotne aplikacije. Ker se v večini primerov uporablja ena entiteta za eno tabelo v podatkovni bazi, je smiselno, da so vse entitete definirane na enem mestu. Vsebuje tudi pogledne objekte za podatke v seznamih vrednosti in aplikacijski modul ADF, ki vsebuje te pogledne objekte.

- **Aplikacija za razvoj sheme podatkovne baze**, če ne razvijamo na obstoječi bazi.

- **Več podsistemov, manjših aplikacij**, ki predstavljajo posamezno funkcionalnost glavne aplikacije. Običajno se za vsak primer uporabe (agilne metode razvoja) razvije en tok opravil. To pomeni, da ima končna aplikacija lahko do sto tokov opravil. Naloga razvjalcev je, da toke opravil smiselno združijo v podsisteme. Vsak podsistem razvija manjša skupina razvijalcev. V projektu modela podsistema so pogledni objekti, ki jih potrebujemo za prikaz podatkov na straneh v tokovih opravil, ter aplikacijski modul ADF. Projekt pogleda *(ang. viewcontroller)* vsebuje tokove opravil ter fragmente strani *(ang. fragments)*. Vsak podsistem vsebuje tudi strani za testiranje, pri katerih mora biti iz imena razvidno, da gre za strani, ki so namenjene testiranju modula in se ne smejo uporabljati v končni aplikaciji.

- **Glavna aplikacija**. To delovno okolje je odvisno od vseh ostalih in ne vsebuje kode. Služi kot vsebovalnik vseh tokov opravil, ki sestavljajo aplikacijo. Na podlagi tega delovnega okolja zgradimo paket s končnico *ear*, ki ga namestimo na strežnik.

Vsak delovni prostor lahko vsebuje več projektov. Če gradimo aplikacijo tipa *Fusion web application (ADF)*, se avtomatsko zgenerirata projekta Model in ViewControler, ki jima pri gradnji večjih aplikacij, spremenimo privzeto ime v *"(ime podsistema)Model"* in *"(ime podsistema)View*". Zaradi konfiguracijskih datotek in podatkovnih vezi moramo paziti, da določimo enoličen paket za oba projekta. Posamezno aplikacijo zapakiramo v knjižnico ADF.

V naslednjem poglavju bomo na primeru predstavili modularen pristop razvoja aplikacij s pomočjo ogrodja ADF in orodja JDeveloper. Uporabljali bomo arhitekturo aplikacije Fusion Web Application.

Poglavje 3

Razvoj aplikacije

V nadaljevanju sledi prikaz razvoja spletne aplikacije, ki služi kot zaledni sistem mobilne aplikacije. Opisane so zahteve aplikacije, rešitve, ki jih ogrodje že ponuja, ter podrobnosti postopka razvoja. Za razvoj aplikacije smo izbrali modularen pristop. Prikazali smo celoten postopek razovoja, od razdelitve projekta v module do priprave namestitvenih profilov in knjižnic za namestitev aplikacije na strežnik. Na koncu smo aplikacijo še testirali, izpostavili prednosti in slabosti izbranega pristopa ter ga primerjali z ostalimi pristopi razvoja aplikacij.

3.1 Tehnologije uporabljene pri razvoju

Za razvoj aplikacije smo uporabili naslednje tehnologije:

- Oracle ADF 12c in ustrezno razvojno okolje JDeveloper,
- strežnika Weblogic 12c ter Glassfish 3.1. Weblogic 12c je integriran v orodje JDeveloper in vsebuje vse potrebne knjižnice za zagon aplikacije ADF. Služi za testiranje aplikacije v fazi razvoja. Knjižnice ADF Essentials smo na strežnik Glassfish namestili sami,
- ADF Essentials,
- podatkovna baza Oracle 11g,
- Subversion SVN za verzioniranje kode.

3.2 Zahteve aplikacije

Razvili smo aplikacijo, ki služi kot zaledni sistem mobilnih aplikacij. Sistem omogoča pregled slik, ki jih uporabniki naložijo preko mobilnih naprav, razdeljevanje uporabnikov v skupine ter pošiljanje obvestil posameznim skupinam ali posameznim uporabnikom. Omogočeno je tudi urejanje podatkov o poslovalnicah oz. objektih za katere želimo, da se prikazujejo na zemljevidu v mobilni aplikaciji.

Primer je mobilna aplikacija za avtomobilsko zavarovanje, kjer zavarovanci plačujejo premijo glede na število prevoženih kilometrov. Prijavljeni uporabniki morajo vsak drugi mesec slikati števce svojega avtomobila in zavarovalnici sporočiti, koliko prevoženih kilometrov ima vozilo. Preko portala se nato preveri, če se vpisani kilometri ujemajo s kilometri na sliki števca.

Aplikacija, ki smo jo razvili omogoča naslednje funkcionalnosti:

- Varnost: Preverjanje pravic uporabnika. Dovoljenja za pregled ali urejanje podatkov. Dovoljenja za pregled le osnovnih podatkov. Dodajanje in urejanje uporabnikov, skupin uporabnikov, ter njihovih pravic.
- Večjezičnost: Prevode aplikacije hranimo v podatkovni bazi. Administratorjem aplikacije je omogočeno, da prevode spreminjajo v času izvajanja aplikacije.
- Pregled in sprejemanje slik, ki jih pošiljajo uporabniki mobilne aplikacije.
- Pregled uporabnikov, prijavljenih v mobilno aplikacijo.
- Možnost grupiranja uporabnikov v skupine. Omogočeno naj bo pošiljanje akcij, obvestil o dogodkih in novostih, skupinam ali posameznim uporabnikom.
- Urejanje poslovalnic.

Zahteve, ki jih ogrodje ADF že rešuje, so naslednje:

- Lokalizacija aplikacije.
- Varnost, ki jo dosežemo z uporabniki, shranjenimi na strežniku Weblogic, ter pravicami, ki jih definiramo v aplikaciji v datoteki *jazn-data.xml*.

3.3 Načrtovanje razvoja

Pri razdeljevanju aplikacije v smiselne module smo opazili, da bo aplikacija vsebovala funkcionalnosti, kot sta preverjanje pravic ter lokalizacija, ki sta prisotni v večini aplikacijah. Zato smo najprej razvili ti dve funkcionalnosti, postavili samostojno aplikacijo, ki omogoča urejajne pravic ter prevodov, in šele nato razvili funkcionalnosti, specifične za portal oziroma zaledni sistem mobilne aplikacije. Splošno aplikacijo, ki vsebuje preverjanje pravic ter lokalizacijo, smo poimenovali kar *ogrodje*.

Za razvoj modulov varnosti in lokalizacije, rešitve, ki jih ponuja ogrodje ADF, ne ustrezajo v celoti zastavljenim zahtevam. Implementirali smo lastno preverjajne pravic ter razširili lokalizacijo, ki jo ponuja ogrodje ADF. Dodatno smo razvili še modul za urejanje in shranjevanje menija v podatkovni bazi. V podatkovni bazi hranimo imena tokov opravil, strani, ter pomožne elemente za meni. Za vsakega uporabnika hranimo dovoljenja za dostop do strani in poganjanja tokov opravil. Tokovi opravil so tudi elementi menija, zato je prikaz elementov menija odvisen od pravic uporabnika.

3.4 Struktura ogrodja

Naše ogrodje mora vsebovati modul za prijavo uporabnika in preverjanje pravic, modul za pridobivanje in urejanje prevodov ter modul za branje in urejanje menija. Na podlagi pravila za delitev aplikacije v module, ki smo ga opisali v poglavju 2.8.2, smo ogrodje razdelili v manjše aplikacije in jim dodelili nivo.

1. Na prvem nivoju je aplikacija, ki vsebuje splošno kodo.

2. Takoj za splošno kodo sledijo aplikacije za preverjajne varnosti, branje prevodov iz podatkovne baze ter branje podatkov o meniju za navigacijo.

3. Aplikacija, ki je odvisna od razredov aplikacije za preverjanje varnosti ter menija, je aplikacija, ki vsebuje predloge izgleda strani ter predlogo toka opravil. Predloga toka opravil vsebuje klic metod, s katerimi preverjamo pravice uporabnika do posameznih tokov opravil. Predloga izgleda strani pa vsebuje fragment, v katerem se prikazuje drevesni meni. Knjižnico, ki jo zgeneriramo iz te aplikacije, bomo uporabili v vseh nadaljnjih modulih, saj bodo vsi tokovi opravil in strani uporabljali predloge, definirane v tem projektu.

4. Na tem nivoju imamo aplikacije za urejanje podatkov preko uporabniškega vmesnika. To so aplikacije za urejajne pravic, prevodov in menija. Vse so odvisne od knjižnic prejšnjih aplikacij. Na tem nivoju so razviti tudi ostali moduli, ki so del neke specifične funkcionalnosti končne aplikacije. Primer so moduli za portal mobilne aplikacije.

5. Končna aplikacija, ki module združi v celoto.

3.5 Splošna koda

Aplikacijo oziroma delovno okolje splošne kode smo poimenovali CommonCode. Vsebuje samo en projekt z istim imenom, v njem pa so razširitveni razredi komponent poslovnega nivoja ADF ter razredi s pogosto uporabljenimi metodami.

Implementirali smo razširitvene razrede za razrede poslovnih komponent, ki smo jih opisali v poglavju 2.2. To so:

- BaoEntityImpl.java, BaoEntityDefImpl.java, BaoEntityCache.java za razrede entitetnega objekta,
- BaoViewDefImpl.java, BaoViewObjectImpl.java, BaoViewRowImpl.java za razrede poglednega objekta,
- BaoApplicationModuleImpl za razrede aplikacijskega modula ADF.

Čeprav v prvi fazi razvoja ne potrebujemo dodatnih funkcionalnosti, je priporočljivo, da v razredih poslovnih komponent ADF uporabljamo lastne razširitvene razrede. V prihodnosti bomo morda potrebovali shranjevati klice poizvedb za vsak pogledni objekt. To bomo storili samo na enem mestu in sicer v metodi *executeQuery()* razširitvenega razreda *ViewObjectImpl.java*.

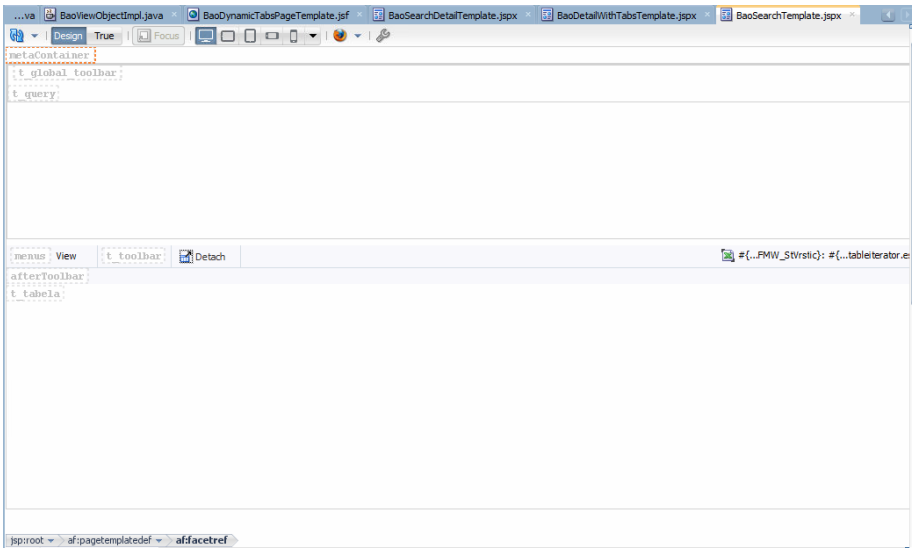
V orodju JDeveloper smo nove razrede nastavili kot privzete razširitvene razrede za poslovne komponente ADF. Pri izdelavi novih komponent, se samodejno uporabijo omenjeni razširitveni razredi.

Razširitvene razrede in razrede, ki vsebujejo splošne metode, smo zapakirali v knjižnico ADF z imenom "adflibCommonCode.jar".

3.6 Splošne komponente kontrolerja in uporabniškega vmesnika

To delovno okolje smo poimenovali CommonUI. Vsebuje predlogo izgleda glavne strani, na kateri je na levem delu zaslona meni. Na desnem delu se v zavihkih prikazujejo tokovi opravil, ki jih je uporabnik pognal. Predlogo strani ter odpiranje tokov opravil iz menija, smo omogočili s pomočjo knjižnice UIShell [13].

Predloge fragmentov strani smo zgradili na podlagi najbolj pogostih razporeditev komponent. Med drugim smo ustvarili predlogo za iskanje in prikaz rezultatov v tabeli, ki je prikazana na sliki 3.1 in predlogo za podroben pregled, za izpis informacij o uporabniku in njegovih sporočilih.



Slika 3.1: Predloga za fragment strani, kjer želimo prikazati komponento za iskanje *(ang. query component)* ter tabelo pod njo.

Projekt vsebuje predlogo toka opravil za preverjajne pravic. Ta vsebuje klic metode, ki preveri, če ima uporabnik pravico do poganjanja toka, še preden se izriše fragment strani. V datoteki adfc-config.xml (nevezan tok opravil) smo definirali zrna Java, ki jih bomo uporabljali skozi celotno aplikacijo. Primer je zrno za dostop do seje uporabnika.

3.7 Aplikacija splošnega modela

Kot je navedeno v podpoglavju 2.8.4, smo entitetne objekte postavili v ločen projekt. Za modul, ki skrbi za prevode, tega nismo storili. Entitetne objekte smo definirali kar v projeku modela za prevode, saj jih ne bomo potrebovali nikjer več. S tem smo dosegli boljšo prenosljivost projekta v druge sisteme, npr. v aplikacije, kjer preverjanja varnosti ne potrebujemo. Če se izkaže, da bomo entitete za prevode potrebovali v drugih aplikacijah, lahko v projektu modela aplikacije za prevode ustvarimo novo knjižnico ADF, kjer bomo zajeli entitete in jih nato uvozili v ostale projekte.

V splošnem projektu modela imamo entitetne objekte, ki ustrezajo tabelam v podatkovni bazi, ter pogledne objekte, ki jih bomo uporabljali kot sezname vrednosti *(ang. list of values)*.

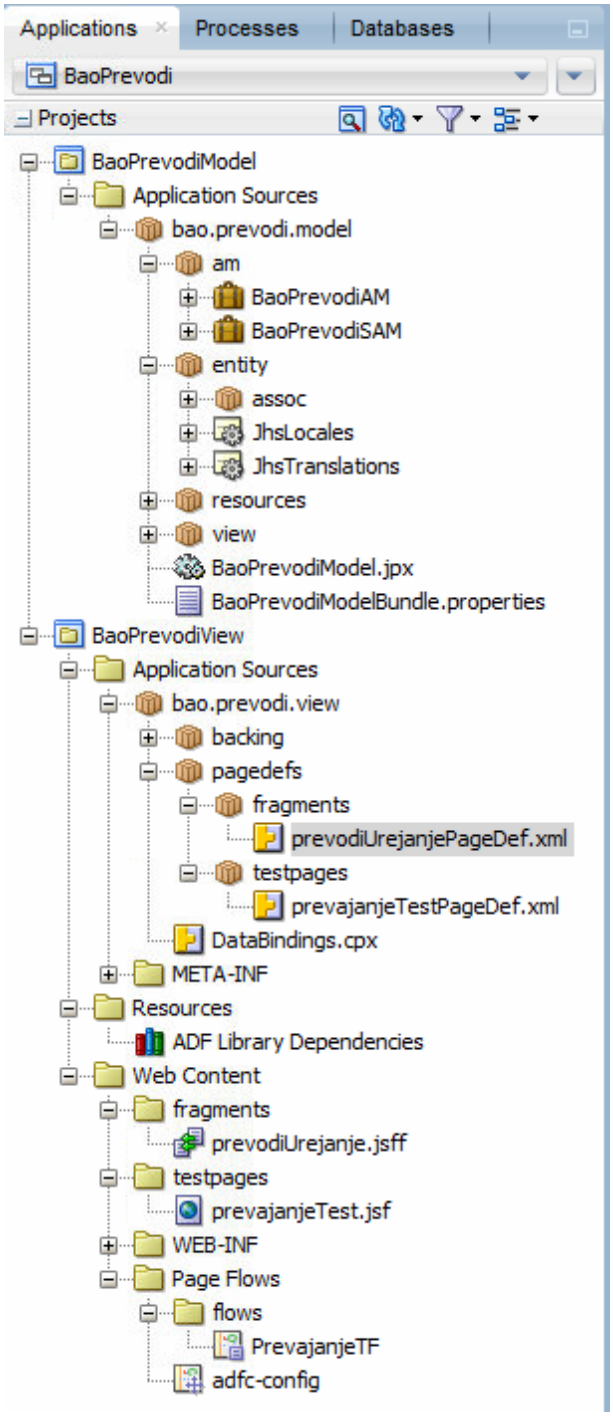
Za modula varnosti in menija smo ustvarili entitete Uporabnik, Skupina, AplikativniModul, UporabnikVSkupini, PravicaDoModula ter PovezavaDoModula. Entitetni objekti so preslikava istoimenskih tabel iz podatkovne baze. Projekt vsebuje tudi asociacije ali entitetne povezave, ki ustrezajo tujim ključem v tabelah. Na vsakem izmed entitetnih objektov smo kasneje definirali tudi ustrezne validatorje, ki jih uporabljamo ob urejanju pravic. Projekt smo zapakirali v knjižnico adflibCommonModel.jar.

3.8 Prevodi

Čeprav nam orodje omogoča hitro lokalizacijo aplikacije, ima ponujena rešitev, ki smo jo opisali v poglavju 2.6, nekaj pomanjkljivosti:

- prevodi so definirani v datotekah. Vsaka sprememba prevoda pomeni novo nameščanje aplikacije;
- orodje ustvari eno ali več datotek s ključi in prevodi. Kadar želimo aplikacijo prevesti v nov jezik, težko določimo kateri ključ pripada določeni oznaki na strani,
- prevode težko ureja nekdo, ki ni razvijalec.

Zgornje pomanjkljivosti smo rešili s prestavitvijo ključev in prevodov v podatkovno bazo. Razvili smo modul za prevode, ki razvijalcu omogoča, da na poglednem objektu ali entiteti omogoči prevajanje z razširitvijo ustreznega razreda. Da bi delo še olajšali, smo razvili razčlenjevalnik *(ang. parser)* XML, ki definicijam entitet in poglednim objektom



Slika 3.2: Prikaz strukture aplikacije (modula) za prevode.

doda ključe. Ključe sestavi na podlagi pravila *ImeEntitete.ImeAtributa* ali *ImePogledenegaObjekta.ImeAtributa*. Za prevajalce smo razvili stran z iskalnikom in tabelo, preko katere lahko urejajo prevode. Če prevoda za nek jezik ni, bo razvijalec na strani videl njegov ključ, ga poiskal v tabeli prevodov in dodal ustrezen prevod za želeni jezik.

3.8.1 Branje prevodov

Kot smo omenili v poglavju 2.8.2, ta projekt spada v drugi nivo, saj je odvisen le od modula za splošno kodo. Ustvarili smo novo delovno okolje tipa ADF Fusion Web Application, katerega struktura je prikazana na sliki 3.2. Projekt modela poleg entitet vsebuje pogledni objekt, s pomočjo katerega preberemo ključe in prevode iz podatkovne baze. Pogledni objekt je vsebovan v aplikacijskem modulu *BaoPrevodiSAM*. V razredu aplikacijskega modula smo implementirali metodo *getPrevodi(String locale)*, ki požene poizvedbo poglednega objekta ter vrne pare (ključ, prevod) za podani jezik. Aplikacijski modul smo definirali kot deljen *(ang. shared)*, kar pomeni, da bodo vsi uporabniki uporabljali isti primerek aplikacijskega modula. Posledično se uporabi tudi ista transakcija v podatkovni bazi.

Aplikacija trenutno podpira slovenski in angleški jezik. V projektu smo ustvarili java razrede *BaoResources.java*, *BaoResources_sl.java* in *BaoResources_en.java*. Razredi razširjajo java razred *ListResourceBunle.java*. V razredu *BaoResources.java* je implementiran klic metode aplikacijskega modula, ki napolni dvodimenzionalno tabelo *contents*. Ker v tem projektu nimamo podatkovnih kontrol in vezi, ne moremo uporabiti knjižnice vezi ADF, ki smo jo opisali v podpoglavju 2.3.1.

Naslednja koda prikazuje, da smo aplikacijski modul ustvarili s pomočjo metode *createRootApplicationModuleHandle()*, ki ji podamo paket in ime razreda aplikacijskega modula, nastavitve tega modula ter razred s konfiguracijo. Po končani operaciji aplikacijski modul, s pomočjo metode *releaseRootApplicationModuleHandle()*, sprostimo nazaj v bazen aplikacijskih modulov. Pri tem se, zaradi ustrezno nastavljene zastavice, stanje modula obdrži. Če izpustimo klic te metode, aplikacijski modul ne bo na voljo ostalim uporabnikom. Lahko pride do uhajanja pomnilnika:

```
public static Object[][] getPrevodi(String LocaleCode) {

        ApplicationModuleHandle handle = null;
        BaoPrevodiSAMImpl am = null;
```

```
        try {
                // Dobimo aplikacijski modul za prevode
                handle = Configuration.createRootApplicationModuleHandle(
                                "bao.prevodi.model.am.BaoPrevodiSAMDefImpl",
                                "BaoPrevodiSAMShared",
                                new BaoDynamicEnvInfoProvider());
                am = (BaoPrevodiSAMImpl)handle.useApplicationModule();
        }catch (Exception e) { ...}

        if (am != null) {
                Map<String, String> prevodiMap = am.getPrevodiJezika(LocaleCode);

                /* Klic metode aplikacijskega modula, ki pridobi prevode
                in jih vrne v zgosceni tabeli */
                Object[][] contents;
                int prevodiMapSize = prevodiMap.size();
                int i = 0;
                if (prevodiMapSize > 0) {
                        contents = new Object[prevodiMapSize][2];
                        for (Map.Entry<String, String> prevod : prevodiMap.entrySet()) {
                                contents[i][0] = prevod.getKey();
                                contents[i][1] = prevod.getValue();
                                i++;
                        }
                } else {
                        contents = new Object[0][0];
                }
                try{
                /* AM sprostimo nazaj v bazen AM. Zaradi zastavice
                'remove=false' se njegovo stanje ohrani. */
                        Configuration.releaseRootApplicationModuleHandle(handle, false);
                }catch(Exception e){ .. }
                return contents;
        }

        return new Object[0][0];
    // return null;
}
```

3.8.2 Uporabniški vmesnik za urejanje prevodov

Uporabniku je omogočeno spreminjanje prevodov preko uporabniškega vmesnika. V projektu modela smo ustvarili nov aplikacijski modul BaoPrevodiAM, ki vsebuje pogledni objekt za urejajne prevodov ter pogledni objekt za prikaz seznama podprtih jezikov. Modul uporablja nastavitve lokalnega aplikacijskega modula, kar pomeni, da bo vsak uporabnik uporabljal svoj primerek tega modula in s tem svojo povezavo na podatkovno bazo.

V projektu pogleda smo ustvarili fragment prevodiUrejanje.jsff. Temu fragmentu pripada vsebovalnik vezi, prevodiUrejajnePageDef.xml, ki se nahaja v paketu bao.prevodi.view.pagedefs.fragments. Orodje JDeveloper vsebovalnik vezi ustvari samodejno ter ga postavi v korenski del paketa. Tega moramo ročno prestaviti v želeni paket, da se v prihodnosti izognemo konfliktom z vsebinki vezi ostalih modulov.

Na strani smo, poleg iskalnika in tabele za urejanje, dodali gumb, ki osveži prevode v aplikaciji. Ker urejanje in pridobivanje prevodov ne spadata v isto bazno transakcijo, saj se uporabljata različna aplikacijska modula in posledično tudi različen medpomnilnik entitet, moramo v poglednem objektu, s katerim beremo prevode aplikacije, ponovno pognati poizvedbo ter osvežiti sveženj oziroma razred s prevodi *(ang. resource bundle)*.

Na sliki 3.2 lahko vidimo, da smo v mapi "test" ustvarili stran PrevodiTest.jsf. Ta stran vsebuje tok opravil za urejanje prevodov ter nam omogoča testiranje funkcionalnosti znotraj modula. V končni aplikaciji s pomočjo dovoljenj onemogočimo dostop do testne strani.

3.9 Varnost

Ogrodje ADF omogoča varnost, ki je tesno povezana z uporabo strežnika Weblogic, za katerega je potrebno plačevati drage licence. Ker bomo našo aplikacijo poganjali na strežniku Glassfish, smo morali implementirati lasten modul za preverjanje pravic uporabnika.

Pravice so zapisane v tabeli entitete AplikativniModul. V njej so zapisani tokovi opravil, strani, dovoljenja do prikaza komponent na strani ter pomožni elementi menija.

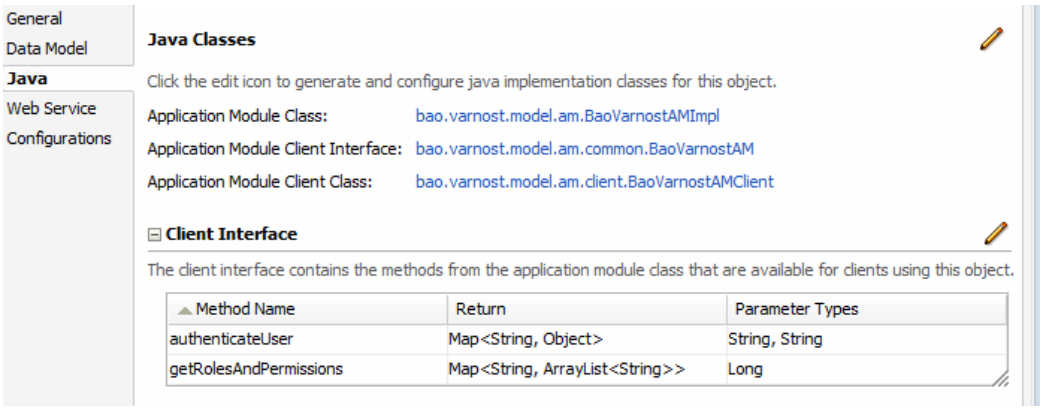
Dostop do strani se omejuje s pomočjo razširitve razreda PagePhaseListener.java, v katerem se v fazi pred izrisom strani preveri, če ima uporabnik omogočen dostop do želene strani. Preverjanje pravic za poganjanje toka opravil se požene v metodi, definirani v

predlogi toka opravil za varnost. Omejitev prikaza posameznih komponent na strani se določa preko atributa Render posamezne komponente.

3.9.1 Prijava uporabnika in preverjanje pravic

Projekt modela

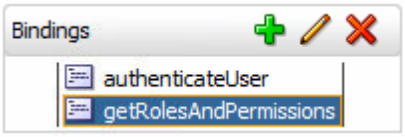
V projektu modela smo ustvarili aplikacijski modul z imenom BaoVarnostAM, ki vsebuje pogledne objekte za avtentikacijo uporabnika ter za pridobitev njegovih pravic. V razredu aplikacijskega modula smo implementirali metodi *authenticateUser()* in *getRolesAndPermissions()*, ki poženeta poizvedbo ustreznih poglednih objektov ter vrneta rezulate v obliki tabele *java.util.Map*. Na sliki 3.3 je prikazano, da sta metodi izpostavljeni odjemalcu, kar pomeni, da lahko do njih dostopamo preko podatkovnih vezi iz uporabniškega vmesnika.



Slika 3.3: Metode aplikacijskega modula, izpostavljene podatkovnim kontrolam v Modelu ADF.

Projekt pogleda

V projektu pogleda smo za prijavo uporabnika ustvarili novo stran, login.jspx. Na strani sta definirana polja za vnos uporabniškega imena in gesla. Vrednosti polj preberemo v upravljalnem zrnu, v katerem nato poženemo metode poslovnih komponent, ki so odjemalcu izpostavljene preko vezi. Ustvarili smo definicijo strani, v kateri so shranjene vezi na metode aplikacijskega modula. Ob prijavi uporabnika se v upravljalnem zrnu požene metoda *dologin()*. Metoda preko vezi prebere metode aplikacijskega modula in



Slika 3.4: Metode aplikacijskega modula, izpostavljene preko vezi Modela ADF.

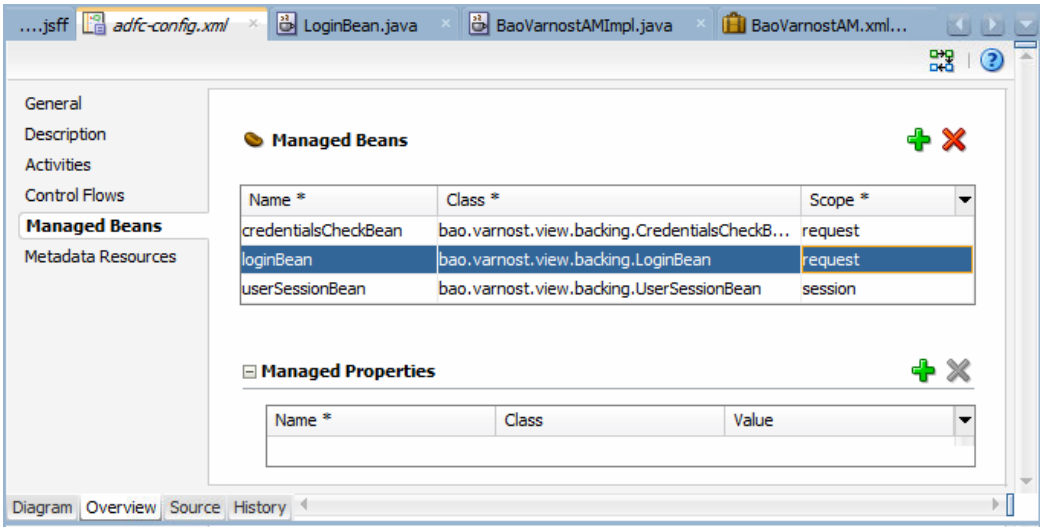
jih izvede. Ker poganjamo metode poslovnega modela v upravljalnem zrnu, smo tukaj lahko uporabili knjižnico vezi ADF *(ang. ADF binding API)*. V spodnji kodi je prikazano poganjanje metode *authenticateUser*, ki je uporabniškemu vmesniku izpostavljena preko podatkovnih vezi, kot prikazuje tudi slika 3.4:

```
DCBindingContainer bind =
(DCBindingContainer)BindingContext.getCurrent().getCurrentBindingsEntry();
OperationBinding opAuthUser =
(OperationBinding)bind.getOperationBinding("authenticateUser");
opAuthUser.getParamsMap().put("username", this.getUporabniskoIme());
opAuthUser.getParamsMap().put("password", this.getGeslo());

opAuthUser.execute();
if (!opAuthUser.getErrors().isEmpty()) {
   //preveri napake
   List errors = opAuthUser.getErrors();
   System.out.println(errors.toString());
}
//vrnjen rezultat
Object resAuthenticateUser = opAuthUser.getResult();
```

Iz slik 3.5 in 3.6 je razvidno, da smo v datoteki adfc-config.xml definirali prijavno stran, login.jspx, ter vsa potrebna zrna za preverjanje pravic uporabnika. Pravice, ki jih vrača metoda *getRolesAndPermissions*, shranimo v sejo uporabnika. Pred odpiranjem strani se v poslušalcu faz preveri, ali ima uporabnik dovoljenje do odpiranja želene strani.

Datoteke adfc-config.xml se bodo v času izvajajna končne aplikacije združile v eno. Strani in zrna, ki smo jih definirali v tem modulu, bodo vidna tudi v končni, glavni aplikaciji. To aplikacijo smo zapakirali v knjižnico z imenom adflibVarnost.jar.



Slika 3.5: Definicija upravljalnih zrn v toku opravil adfc-config.xml.

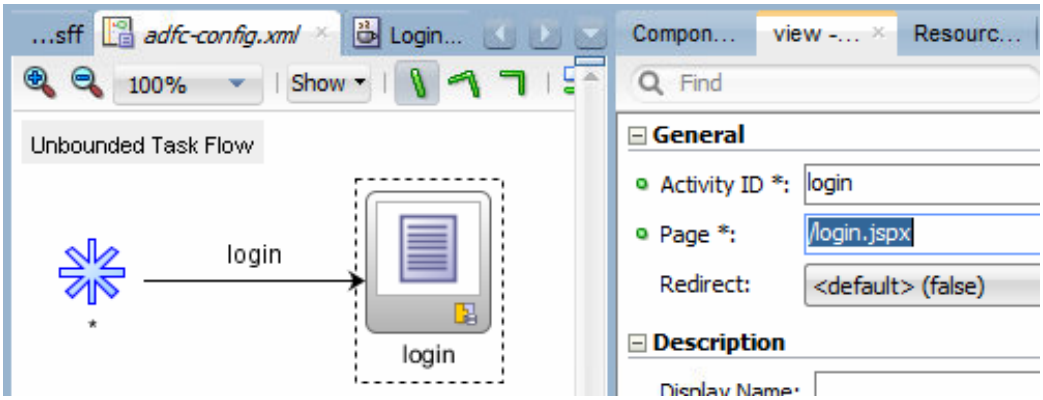
3.9.2 Urejanje uporabnikov in pravic

Razvili smo novo aplikacijo, ki vsebuje svoj projekt pogleda in projekt modela. V projekt modela smo uvozili knjižnice, ki vsebujejo splošen model z entitetami, splošno kodo in modul za branje prevodov. Ustvarili smo nov aplikacijski modul s poglednimi objekti za prikaz in urejanje podatkov o uporabnikih, skupinah ter pravicah. V projekt pogleda smo uvozili knjižnico adflibCommonUI.jar, saj potrebujemo predlogo strani za iskanje ter predlogo toka opravil, s katerim omogočimo omejitev dostopa glede na pravice. Aplikacijo smo zapakirali v knjižnico adflibPraviceUrejanje.jar ter jo uvozili v projekt ogrodja.

3.10 Aplikacija ogrodja in uvoz knjižnic

Ogrodje je, prav tako kot moduli, aplikacija tipa Oracle Fusion Web Application. Aplikacija, ki predstavlja ogrodje, vsebuje zelo malo elementov. V projekt pogleda smo uvozili vse projekte, ki smo jih omenili v prejšnjih poglavjih. To so knjižnjice: adflibBaoCommonCode.jar, adflibBaoCommonUI.jar, adflibBaoVarnost.jar, adflibBaoPrevodi.jar adflibBaoVarnostUrejanje.jar, adflibBaoPrevodiUrejanje.jar.

Dodatno smo ustvarili novo stran s končnico jsf in jo poimenovali *home.jsf*. Stran temelji na predlogi izgleda, ki vsebuje meni za navigacijo, zavihke, v katerih poganjamo



Slika 3.6: Definicija strani login.jspx v toku opravil adfc-config.xml.

tokove opravil, ter informacijo o prijavljenem uporabniku. V neomejen tok opravil, adfc-config.xml, smo dodali stran *home.jsf* ter iz nadomestnega elementa *(ang. wildcard)*, zvezdice, definirali akcijo z nazivom "home". Nadomestni element se uporabi, kadar želimo akcijo klicati z vseh strani, definiranih v toku opravil.

Ob zagonu aplikacije se bo vsebina datotek adfc-config.xml iz ostalih podsistemov združila v eno. Nevezan tok opravil bo vseboval še stran za prijavo, akcijo, ki vodi do nje, ter vsa upravljalna zrna, ki smo jih definirali v posameznih modulih. Pred dostopom do določene strani se najprej preveri, če ima uporabnik pravico do odpiranja želene strani. Ko uporabnik ni prijavljen, nima pravic do dostopa na domačo stran, zato ga aplikacija preusmeri na prijavno stran.

Za namestitev aplikacije na strežnik najprej ustvarimo namestitveni profil *(ang. deploy profile)* na projektu pogleda in nato aplikacijo zapakiramo v datoteko s končnico *ear*. V nastavitvah projekta pogleda določamo ime aplikacije ter prikaz imena v naslovni vrstici brskalnika.

Posamezne module bi lahko na strežnik namestili kot deljene knjižnice ADF *(ang. ADF Shared Library)*. S tem ne bi onemogočili delovanja celotne aplikacije, ampak le modula, ki ga nameščamo. Pri tem pristopu moramo v datoteki weblogic.xml ročno definirati odvisnosti med moduli ter paziti na ustreznost namestitvenih profilov. Knjižnice modulov smo namestili skupaj z ogrodjem, saj gre za preprosto aplikacijo z velikimi odvisnostmi med moduli.

3.11 Moduli aplikacije portala

Do tega dela smo postavili delujoče ogrodje, ki omogoča prijavo uporabnika, urejanje uporabnikov in pravic ter urejanje prevodov. V naslednjih korakih smo razvili module, specifične za portal za pregled uporabnikov, prijavljenih v mobilno aplikacijo. Podrobnosti razvoja posameznih modulov ne bomo opisovali, saj je postopek enak tistemu, ki smo ga uporabili pri razvoju modulov za urejanje pravic in prevodov.

Aplikacijo portala smo ločili v module za: preveranje slik, urejanje poslovalnic ter pregled uporabnikov in njihovih sporočil.

Ustvarili smo novo aplikacijo splošnega poslovnega modela, ki vsebuje entitetne objekte za nove module. Ustvarili smo tudi nov projekt za spreminjanje izgleda komponentam ogrodja ADF Faces. V vsak modul smo uvozili knjižnice splošne kode, splošnega modela portala, knjižnico za branje prevodov ter knjižnico, ki vsebuje tok opravil za preverjajne pravic. Module smo zapakirali v knjižnice ADF in jih uvozili v ogrodje. Zaradi preverjanja pravic smo vse module oziroma imena tokov opravil morali zapisati tudi v bazi, da smo nato lahko definirali pravice posameznega uporabinka. Če tega ne storimo, se modul oziroma, natančneje, tok opravil ne bo prikazoval niti na meniju.

3.11.1 Oblikovanje komponent *(ang. ADF Skinning)*

V posameznih modulih smo morali komponentam ogrodja ADF Faces spremeniti izgled. To smo storili z novo preobleko *(ang. skin)* aplikacije. Za oblikovanje koponente tabele smo definirali nov razred *"IaugMSTableG"*, s pomočjo katerega tabeli definiramo stil. Če bi tabelo oblikovali kar preko izbirnika *af|table* za določanje stila tabele, bi obliko spremenili vsem tabelam v aplikaciji. Z razredom pred zbirnikom omogočimo, da bo komponenta prevzela obliko, definirano v novi preobleki le, če bomo v atribut Styleclass nadrejene komponente vpisali ime razreda. Rezultat je viden na sliki 3.12 v poglavju 3.12. V naslednji kodi je prikazano oblikovanje celice tabele, kjer smo celicam določili debelejšo obrobo:

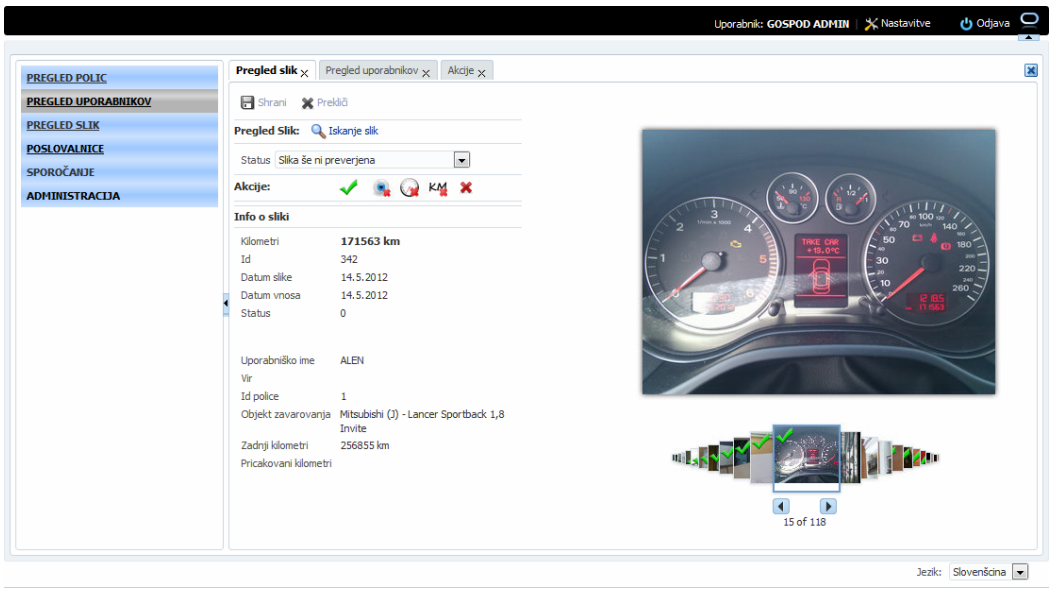
```
.IaugMsTableG af|table::data-row af|column::data-cell{
  border-top: 5px solid;
  border-bottom: 5px solid;
  border-color: white;
  padding: 0px;
  color:rgb(82,82,82);
```

```
}
```

```
<af:table value="#{bindings.IosSkupineVAkciji1.collectionModel}"
          var="row"
          rows="#{bindings.IosSkupineVAkciji1.rangeSize}"
          ...
          styleClass="IaugMsTableG" width="298">
          <af:column>
             <!-- vsebina stolpca -->
          </af:column>
</af:table>
```

3.12 Testiranje aplikacije

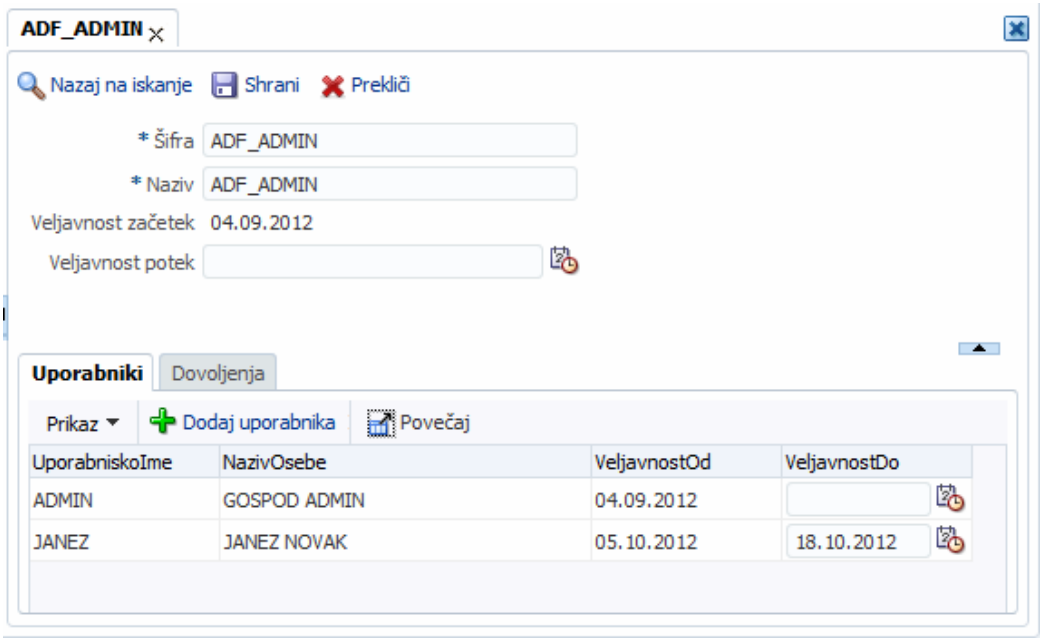
Ko aplikacijo poženemo, se najprej pojavi prijavno okno. Uporabnik vpiše svoje uporabniško ime in geslo ter klikne na gumb prijava. V ozadju se pokličeta metodi *authenticateUser()* in če avtentikacija uporabnika uspe, tudi metoda *getUserPermissions()*. Uporabnik je nato preusmerjen na stran home.jsf. Na sliki 3.7 je prikazan izgled aplikacije. Stran vsebuje meni, podatek o prijavljenem uporabniku, gumb za odjavo in zavihke, ki predstavljajo odprte tokove opravil.



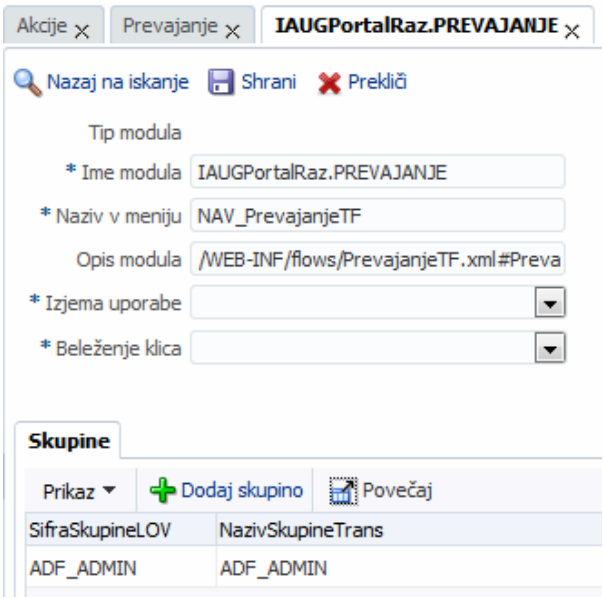
Slika 3.7: Izgled aplikacije. Na levi se nahaja meni, na vrhu podatki o uporabniku ter gumb za odjavo in zavihki, ki predstavljajo odprte tokove opravil.

3.12.1 Urejanje dovoljenj

Sliki 3.8 in 3.9 prikazujeta urejanje dovoljenj in skupin, ki spadata v modul varnosti. Prikazani so uporabniki skupine ADF_ADMIN ter skupine, ki imajo dovoljenje do poganjanja toka opravil za urejanje prevodov. Uporabniki, ki v to skupino ne spadajo, na meniju ne bodo videli gumba za urejanje prevodov. Dovoljenje do odpiranja toka opravil je v podatkovni bazi zapisano kot enoličen niz, ki se v aplikaciji uporabi kot pot za odpiranje želenega toka opravil.



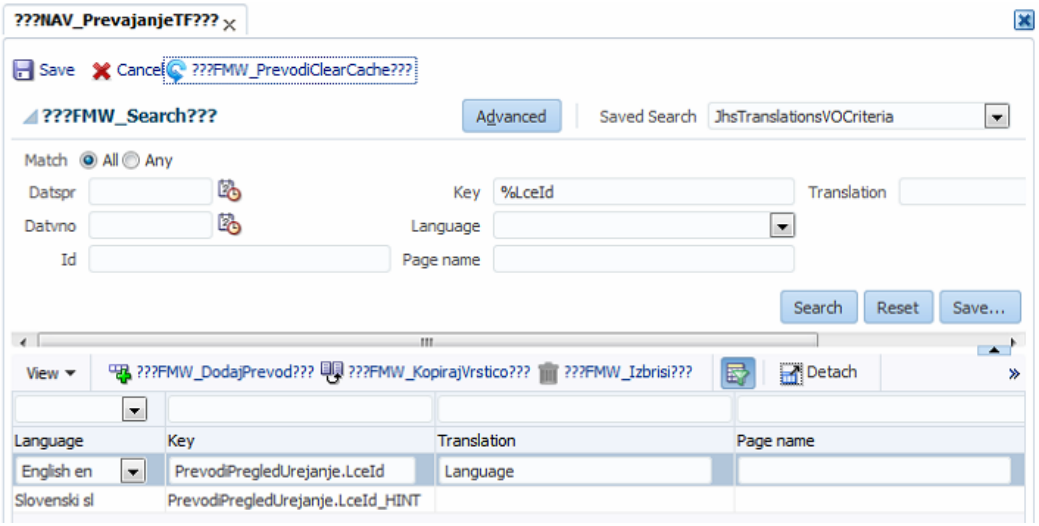
Slika 3.8: Modul za varnost. Urejanje uporabnikov v skupini.



Slika 3.9: Modul za varnost. Dovoljenje in pripadajoče skupine. Skupine, ki imajo dovoljenje do odpiranja toka opravil za prevajanje aplikacije.

3.12.2 Urejanje prevodov

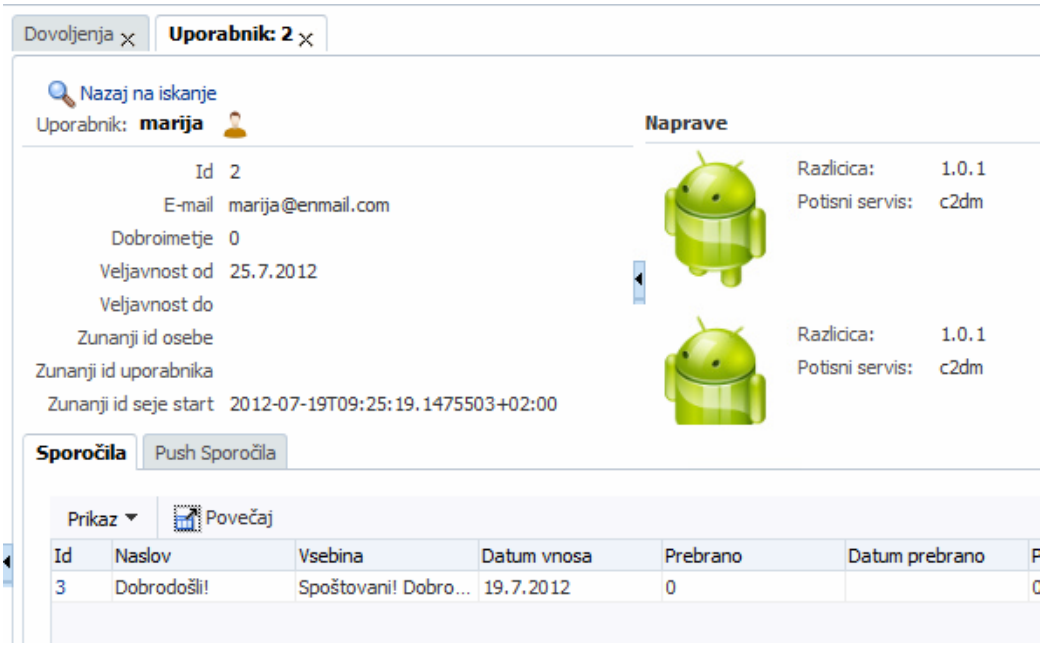
Na sliki 3.10 je prikazano urejanje prevodov. Lahko opazimo, da so atributi poglednega objekta, preko katega urejamo prevode, že prevedeni v angleški jezik, medtem ko nekatera imena gumbov niso. Tudi ime toka opravil, to je *NAVPrevajanjeTF*, ni prevedeno v angleški jezik.



Slika 3.10: Urejanje prevodov in prikaz oznak, ki niso še prevedene.

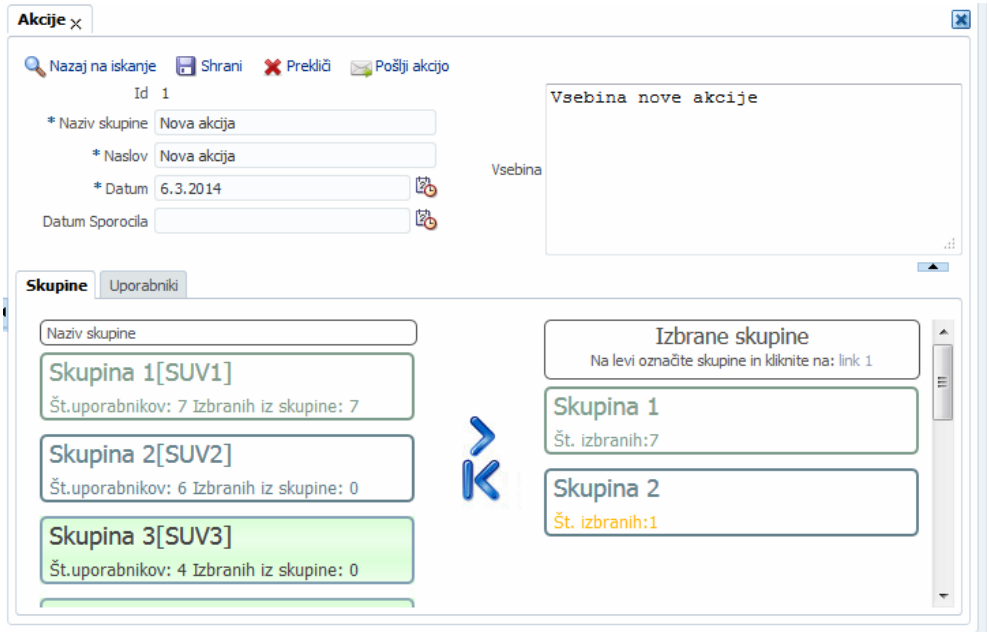
3.12.3 Portal - zaledni sistem mobilne aplikacije

Slika 3.11 prikazuje pregled uporabnikov, ki so prijavljeni v mobilno aplikacijo, pregled naprav, preko katerih uporabnik dostopa do aplikacije, ter sporočil, ki so bila poslana uporabniku.

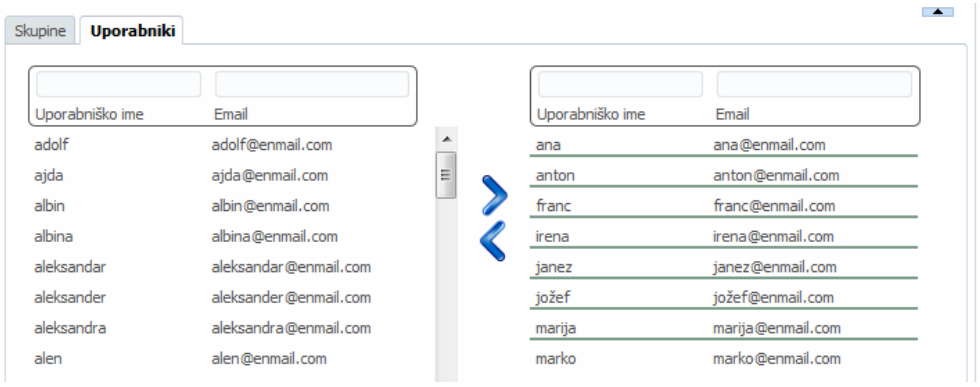


Slika 3.11: Pregled uporabnika, prijavljenega v mobilno aplikacijo.

Na slikah 3.12 in 3.13 je prikazano urejanje akcij. To so sporočila ali obvestila, ki jih pošiljamo uporabnikom mobilne aplikacije. Na sliki 3.12 lahko vidimo, da je akcija namenjena uporabnikom skupine z nazivom *"Skupina 1"* ter uporabniku, ki slučajno spada v skupino z nazivom *"Skupina 2"*. Za prikaz skupin smo uporabili komponento tabele ogrodja ADF Faces in ji s pomočjo orodja ADF Skinning spremenili izgled. Na sliki 3.13 je v levi tabeli prikazan seznam uporabnikov, ki jih lahko dodamo v akcijo, na desni pa seznam uporabnikov, ki so že zajeti v akciji. Iz črte pod imenom uporabnika lahko vidimo, s katero skupino smo ga vključili v akcijo. Uporabnik "marko" je bil dodan naknadno, zato njegova vrstica ni obarvana z ustrezno barvo skupine.



Slika 3.12: Urejanje nezaključene akcije. Dodajanje skupin.



Slika 3.13: Urejanje nezaključene akcije. Dodajanje skupin uporabnikov.

3.13 Primerjava pristopov razvoja

V poglavju 2.8 smo omenili, da je izbira pristopa odvisna predvsem od velikosti aplikacije, števila razvijalcev, iterakcije med posameznimi aplikacijami ter načina nalaganja aplikacije na strežnik.

Pri monolotičnem pristopu imamo samo eno delovno okolje. Omogoča hiter in enosta-

ven razvoj osnovnih aplikacij. Testiranje končne aplikacije poteka hitro, vse funkcionalnosti in datoteke pa so na enem mestu. Razvijalcu ni potrebno razmišljati o odvisnostih med posameznimi projekti. Slabost tega pristopa se pokaže predvsem takrat, ko postane aplikacija bolj obsežna in na njej dela več razivjalcev. Ti se pogosto soočajo s težavami, kot so reševanje konfliktov v sistemu za nadzor različic. Zaradi deklarativnega razvoja orodje JDeveloper veliko datotek ter podatkov v njih zapiše samodejno, brez vednosti razvijalca. Tak primer je datoteka vezi DataBindings.cpx, ki vsebuje podatke o straneh, tokovih opravil ter pripadajočih vsebnikov vezi. To sicer niti ni tako velik problem, če vsi razvijalci dobro poznajo razvojno ogrodje in vedo, na katere datoteke morajo paziti.

Z večanjem kompleksnosti aplikacije se spreminja tudi velikost delovnega okolja. Ta lahko vsebuje veliko datotek, kar lahko povzroči nepreglednost ter upočasnjuje razvojno orodje. Kadar aplikacijo razvija več razvijalcev in ta ni preveč obsežna, lahko uporabimo monolitični pristop, eno delovno okolje, kjer uporabimo večje število projektov modela in pogleda. Vsak razvijalec lahko nemoteno razvija v svojem projektu in uporablja funkcionalnosti ostalih projektov. Pri tem pristopu je potrebno že razmišljati o delitvi aplikacije v smiselne podprojekte in paziti na odvisnosti med njimi. V kolikor aplikacija postane preveč obsežna, lahko posamezne projekte postavimo v svoje delovno okolje in aplikacijo naprej razvijamo modularno.

Prednost modularnega pristopa je predvsem omogočanje večuporabnosti posameznih modulov. Sili k razvoju manjših samostojnih aplikacij, ki jih lahko združujemo v poljubne večje aplikacije. To seveda prinaša nekaj več dela. V začetku je to zaradi definicije strukture aplikacije, kasneje pa zaradi same organizacije knjižnic ter skrbi, da so zadnje različice le-teh vedno dostopne razvijalcem. Vsak modul mora imeti tudi svoj sistem za vodenje različic kode.

V aplikacijah, kjer je veliko interakcije med posameznimi poslovnimi procesi, lahko projekte znotraj modula ločujemo na več projektov modela in pogleda. Za vsak projekt ustvarimo svojo knjižnico ADF. Pristop je opisan v poglavju 2.8.2, delno pa smo ga prikazali pri razvijanju modula za prevode. Poslovne komponente tega modula smo zapakirali v ločeno knjižnico, ki smo jo kasneje uvozili v ostale projekte, kjer smo potrebovali prevajanje atributov poglednih in entitetnih objektov. Knjižnico smo uvozili tudi v projekt pogleda za urejanje prevodov. V projekt modela ostalih aplikacij ne smemo uvažati knjižnice, ki vsebujejo elemente uporabniškega nivoja. Orodje JDeveloper v takem primeru samodejno ustvari določene mape in datoteke, ki so v projektu

modela nepotrebne in včasih celo odvečne, saj lahko onemogočijo delovanje aplikacije in jih moramo ročno odstraniti.

Za razvoj modulov portala smo uporabili pristop, kjer projekte posameznih modulov zapakiramo v eno samo knjižnico ADF. Ta pristop smo izbrali, ker aplikacija ni obsežna in poslovnih komponent nekega modula ne potrebujemo v drugih projektih. Diagram take aplikacije je prikazan na sliki 2.9 v poglavju 2.8.4.

Pri modularnem pristopu se veliko časa porabi za grajenje knjižnjic, menjavo delovnega okolja oz. aplikacij ter testiranje končne aplikacije. Primer, ko spreminjamo izgled komponentam, moramo to najprej storiti v aplikaciji splošnega izgleda, to aplikacijo zapakirati v knjižnico ADF, jo uvoziti v modul, kjer želimo preveriti spremenjeno komponento, ter ta modul pognati. Poganjanje aplikacije ADF v povprečju traja eno minuto.

Kadar delamo in testiramo spremembe znotraj posameznega modula, lahko uporabimo tehnologijo fast-swap. Ta nam omogoča, da spremembo razredov ali strani vidimo že z osvežitvijo strani v spletnem brskalniku. Razvoj posameznih modulov se zaradi tega zelo pohitri. Tehnologije fast-swap ne moremo uporabiti v primeru, kadar želimo spremembe modula testirati v končni aplikaciji, saj v osveževanje ne zajame razredov v knjižnicah.

V ta namen obstajajo tudi zunanja orodja, kot je orodje JRebel, ki omogoča, da spremembe v modulu testiramo v končni aplikaciji le z osveževanjem strani v spletnem brskalniku. Orodje upravlja tudi s konfiguraciskimi datotekami ogrodja ADF, kot so definicije strani s podatkovnimi vezmi. Razvoj aplikacije je z uporabo orodja JRebel bistveno hitrejši, edini problem pa je, da trenutno ne zna najbolje sodelovati z razhroščevalnikom orodja JDeveloper.

Poglavje 4

Zaključek

Pri razvoju aplikacij ni pomembna samo izbira ogrodja, ampak tudi izbira pravega pristopa razvoja aplikacije z izbranim ogrodjem. Osnove poznavanja ogrodja ne zadoščajo za razvoj obsežnih aplikacij. Za ogrodje Oracle ADF obstaja na spletu veliko forumov in blogov, ki vsebujejo primere za hitrejše in lažje učenje. Slabost tega je, da lahko, poleg ustreznih primerov, na spletu najdemo veliko takih, ki nas vodijo k napačni uporabi ogrodja. Zato je pomembno, da razvijalec dobro preuči funkcionalnosti, ki jih ponuja ogrodje. Oracle na tem področju ponuja ogromno dokumentacije, ki pa je za razvijalca, ki šele spoznava ogrodje, preobsežna in preveč podrobna. Dokumentacije o ogrodju je zelo veliko, prebijanje skozi njo pa je lahko za razvijalca mučno in dolgotrajno. V zadnjih letih je zato bilo spisane kar nekaj literature, s pomočjo katere dobi razvijalec hiter vpogled v načine razvoja ter uporabo dobrih praks z ogrodjem.

V diplomski nalogi smo primerjali v literaturi omenjene pristope k razvoju aplikacij z ogrodjem ADF. Na primeru portala oziroma zalednega sistema mobilne aplikacije smo prikazali modularen razvoj obsežnejših aplikacij.

Aplikacija, ki smo jo razvili, je preprosta in vsebuje malo strani. Za razvoj bi lahko uporabili enostavnejši, monolitični pristop, vendar tega nismo storili, saj smo želeli preveriti pristop, namenjen razvoju obsežnejših aplikacij. Na podlagi razvite aplikacije smo prikazali prednosti in slabosti modularnega pristopa ter izpostavili težave, ki se lahko pojavljajo tekom razvoja aplikacije. Tem se bomo lahko izognili ob razvoju obsežnejših aplikacij. V praktičnem delu naloge smo poleg izbranega pristopa prikazali, kako pomembno je dobro poznavanje ogrodja, ki ga uporabljamo. Kjub temu, da ogrodje ADF nudi razvoj javanskih aplikacij z upoštevanjem dobrih praks, lahko z napačnim pristopom

onemogočimo določene funkcionalnosti, ki jih ogrodje ponuja.

Veliko podjetij zmotno pričakuje, da ogrodje omogoča gradnjo bogatih spletnih aplikacij, kjer lahko vse funkcionalnosti, predvsem v uporabniškem vmesniku, prilagajamo željam posameznih naročnikov. Ne zavedajo se, da se za razvoj še vedno uporablja ogrodje s tehnologijami, ki niso vedno najsodobnejše. Uporaba ogrodja ima veliko prednosti, vendar tudi omejitve, ki jih moramo sprejeti, če želimo doseči hiter in učinkovit razvoj.

Slike

Literatura

[1] Sten E. Vesterli. *Oracle ADF Enterprise Application Development - Made Simple*. Packt Publishing, 2011.

[2] Sten E. Vesterli. *Developing Web Applications with Oracle ADF Essentials*. Packt Publishing, 2013.

[3] J. Purushothaman. *Oracle ADF Real World Developer's Guide*. Packt Publishing, 2012.

[4] Patric Keegan. *Oracle Fusion Middleware Understanding Oracle Application Development Framework 12c(12.1.2)*. Oracle, 2013.

[5] Andrejus Baranovskis. ADF BC Passivation/Activation and SQL Execution Tuning, dostopno na: https://http://andrejusb.blogspot.com/2012/08/adf-bc-passivationactivation-and-sql.html

[6] Vinod Krishnan. *Oracle ADF 11gR2 Development Beginner's Guide*. Packt Publishing, 2013.

[7] (2011) An Oracle white paper, Oracle Application Development, Framework Overview, dostopno na: http://www.oracle.com/technetwork/developer-tools/adf/adf-11-overview-1-129504.pdf

[8] Ralph Gordon. *Oracle® Fusion Middleware, Fusion Developer's Guide for Oracle Application Development Framework 11g Release 2 (11.1.2.1.0)*. Oracle, 2011.

[9] What's new in JSF 2.2?, dostopno na: http://jdevelopment.nl/jsf-22/#730

[10] Pon Lee Tan. *Oracle® Fusion Middleware Developing Web User Interfaces with Oracle ADF Faces 12c(12.1.2)*. Oracle, 2013.

[11] (2012) Joab Jackson. Oracle releases free ADF Essentials, dostopno na: http://www.oracle.com/technetwork/developer-tools/adf/adf-11-overview-1-129504.pdf

[12] Oracle ADF Essentials Overview and Frequently Asked Questions, dostopno na: http://www.oracle.com/technetwork/developer-tools/adf/overview/adfessentialsfaq-1837249.pdf

[13] Core ADF11: UIShell with Dynamic Tabs, dostopno na: https://blogs.oracle.com/jheadstart/entry/core_adf11_uishell_with_dynamic